Manuel d'utilisation de XLOGO :

Le Coq Loïc

25 mars 2007

Table des matières

1 Présentation de l'interface : 3
- 1.1 Fenêtre principale . . . 3
- 1.2 L'éditeur de procédures . . . 4

2 Options des menus : 5
- 2.1 Menu « Fichier » . . . 5
- 2.2 Menu « Edition » . . . 5
- 2.3 Menu « Options » . . . 5
- 2.4 Menu « Aide » . . . 6

3 Conventions adoptées dans XLOGO 8
- 3.1 Commandes et interprétation . . . 8
- 3.2 Procédures . . . 8
- 3.3 Le caractère spécial « \ » . . . 9
- 3.4 Règles concernant les majuscules et minuscules . . . 9
- 3.5 Opérateurs et syntaxe . . . 10
- 3.6 Les couleurs . . . 11

4 Liste des primitives 12
- 4.1 Déplacement de la tortue, gestion du crayon et des couleurs . . . 12
- 4.2 Affichage du texte dans la zone d'historique . . . 15
- 4.3 Opérations arithmétiques et logiques . . . 17
- 4.4 Opérations sur les listes et les mots . . . 19
- 4.5 Booléens . . . 21
- 4.6 Effectuer un test à l'aide de la primitive `si` . . . 22
- 4.7 Gestion des procédures et des variables . . . 22
  - 4.7.1 Les procédures . . . 22
  - 4.7.2 Notion de variables . . . 23
  - 4.7.3 La primitive '`trace`' . . . 23
- 4.8 Gestion des Fichiers . . . 25
- 4.9 Fonction avancée de remplissage : . . . 28
- 4.10 Les instructions de saut . . . 30
- 4.11 Le mode multitortues . . . 30
- 4.12 Jouer de la musique . . . 31
- 4.13 Boucles : . . . 33
  - 4.13.1 Une boucle avec `repete` . . . 33
  - 4.13.2 Une boucle avec `repetepour` . . . 33
  - 4.13.3 Une boucle avec `tantque` . . . 34
- 4.14 Intercepter des actions de l'utilisateur . . . 35

4.14.1 Interaction avec le clavier . . . 35
4.14.2 Quelques exemples d'utilisation : . . . 35
4.14.3 Intercepter certains événements provenant de la souris . . . 36
4.14.4 Quelques exemples d'utilisation . . . 36
4.15 Gestion du temps . . . 38
4.16 Se servir du réseau avec XLogo . . . 39
4.16.1 Le réseau : comment ça marche ? . . . 39
4.16.2 Primitives orientées réesau . . . 39

5 Exemples de programmes **41**
5.1 Dessiner des maisons . . . 41
5.2 Dessiner un rectangle plein . . . 43
5.3 Factorielle . . . 43
5.4 Le flocon (Merci à Georges Noël) . . . 43
5.5 Un peu d'écriture... . . . 45
5.6 Et de conjugaison... . . . 46
5.6.1 Première version . . . 46
5.6.2 Deuxième mouture . . . 46
5.6.3 Ou alors : Un peu de récursivité ! . . . 46
5.7 Avec les couleurs . . . 47
5.7.1 Introduction . . . 47
5.7.2 Passons à la pratique . . . 47
5.7.3 Et si on la voyait en négatif ? ? . . . 48
5.8 Un bel exemple d'utilisation des listes (Merci Olivier SC) . . . 49
5.9 Une belle rosace . . . 50

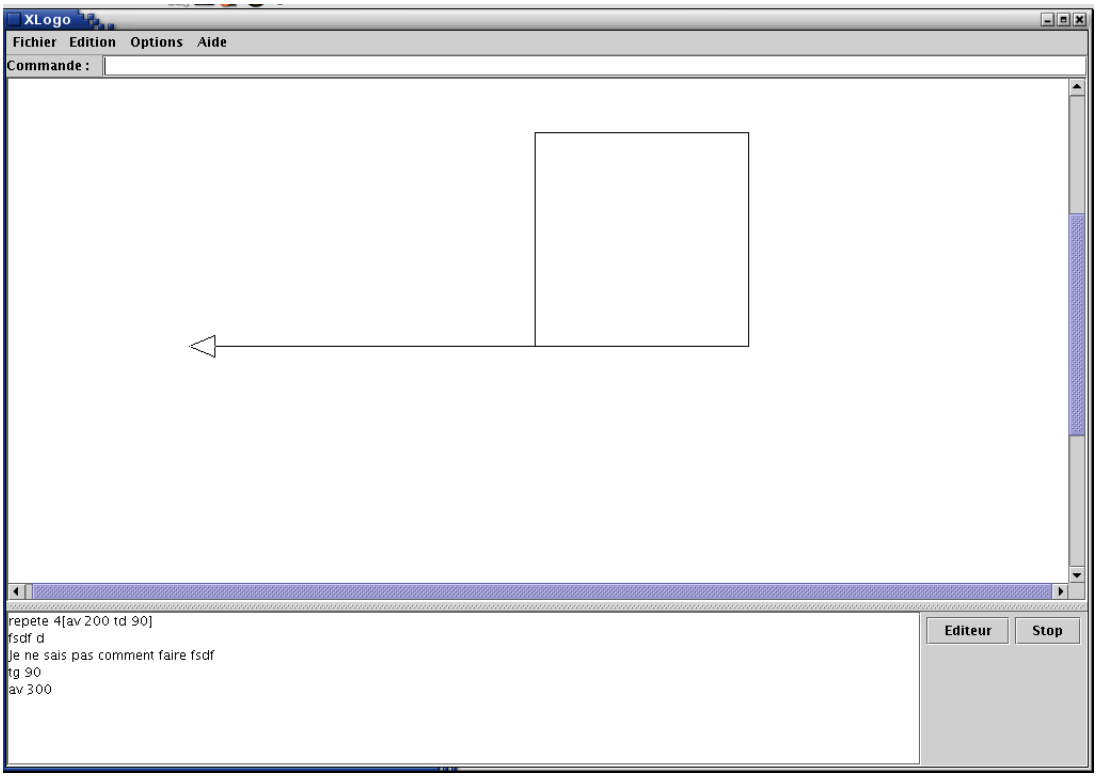
6 Désinstallation-mise à jour **51**
6.1 Désinstallation . . . 51
6.2 Mise à jour . . . 51

7 Foire aux questions - Astuces - trucs à connaître **52**
7.1 J'ai beau effacer une procédure dans l'éditeur, elle réapparaît tout le temps ! . . . 52
7.2 J'utilise la version en espéranto mais je ne peux écrire les caractères spéciaux ! . . . 52
7.3 Dans l'onglet Son de la boîte de dialogue Préférences, aucun instrument n'est disponible. 52
7.4 J'ai des problèmes de rafraichissement d'écran lorsque la tortue dessine ! . . . 52
7.5 Comment faire pour taper rapidement une commande déjà utilisée au préalable ? . . . 53
7.6 Comment peut-on vous aider ? . . . 53

Chapitre 1

Présentation de l'interface :

1.1 Fenêtre principale



- En haut, les traditionnels menus `Fichier Edition Options et Aide`
- Juste en dessous, la ligne de commande qui permet de saisir les instructions logo.
- Au centre, la zone de dessin.
- En bas, la zone « historique » qui rappelle toutes les dernières commandes tapées et les réponses associées. Pour rappeler rapidement une instruction déjà tapée, il y a deux solutions : ou bien vous cliquez sur l'ancienne instruction dans l'historique, ou bien vous appuyez plusieurs fois sur la flèche du haut jusqu'à ce que l'instruction désirée apparaisse. Les deux flèches haut et bas permettent en fait de se déplacer dans toute l'historique des commandes tapées précédemment (Très pratique).
- A la droite de l'historique, deux boutons : STOP et EDITEUR. STOP interrompt toute exécution en cours et EDITEUR permet d'ouvrir l'éditeur de procédures.

1.2 L'éditeur de procédures

Pour ouvrir l'éditeur, trois possibilités :

- Taper `ed` dans la ligne de commandes. L'éditeur s'ouvrira alors avec toutes les procédures déjà définies. Si vous ne souhaitez éditer que certaines procédures particulières, taper alors : `ed [procedure_1 procedure_2 ...`
- Appuyer sur le bouton Editeur de la fenêtre principale.
- Utiliser le raccourci clavier Alt+E

Voici les différents boutons que vous trouverez dans l'éditeur :

Sauve les modifications apportés au contenu de l'éditeur puis ferme celui-ci. C'est sur ce bouton qu'il faut appuyer à chaque fois que vous voulez enregistrer les nouvelles procédures tapées. Si vous le préférez, vous pouvez utiliser le raccourci clavier ALT+Q.

Quitte l'éditeur en n'enregistrant aucune des modifications apportées à celui-ci. On peut également utiliser le raccourci ALT+C.

Imprime le contenu de l'éditeur.

Copie le texte sélectionné dans le presse-papiers

Coupe le texte sélectionné dans le presse-papiers

Colle le texte sélectionné dans le presse-papiers

IMPORTANT :

- Cela ne sert à rien d'appuyer sur la croix en haut à droite pour fermer la fenêtre ! Seuls les deux premiers boutons vous permettent de quitter l'éditeur.
- Pour effacer une ou toutes les procédures indésirables, utiliser les primitives `efn` et `efns`.

Chapitre 2

Options des menus :

2.1 Menu « Fichier »

- Fichier –> Nouveau : Détruit l'ensemble des procédures et variables définies pour créer ainsi un nouvel espace de travail.
- Fichier –> Ouvrir : ouvre un fichier logo précédemment enregistré.
- Fichier –> Enregistrer : enregistre les procédures du fichier en cours.
- Fichier –> Enregistrer sous ... : enregistre les procédures en cours sous un nom précis.
- Fichier –> Quitter : quitte l'application XLOGO.
- Fichier –> Capturer l'image –> Enregistrer l'image sous... : permet d'enregistrer l'image sous le format jpg ou png. Si vous souhaitez sélectionner seulement une partie de l'image, vous avez la possibilité de définir un cadre en cliquant successivement deux fois pour définir deux coins d'une même diagonale du rectangle de sélection.
- Fichier –> Capturer l'image –> Imprimer l'image : permet d'imprimer l'image. De la même façon que précédemment, vous pouvez sélectionner une zone à imprimer.
- Fichier –> Capturer l'image –> Copier l'image dans le presse-papier : Permet d'envoyer l'image dans le presse-papier système. De même que pour l'impression et l'enregistrement, vous pouvez ne sélectionner qu'une zone de l'image. Cette fonctionnalité fonctionne très bien sous les environnements de type Windows. En revanche, elle ne marche pas sous Linux (Le presse-papier n'a pas le même type de fonctionnement). Non testé sous Mac.

2.2 Menu « Edition »

- Edition –> Copier : copie le texte sélectionné dans le presse-papiers.
- Edition –> Couper : coupe le texte sélectionné dans le presse-papiers.
- Edition –> Coller : colle le texte contenu dans le presse-papiers dans la ligne de commande.

2.3 Menu « Options »

- Options –> Choisir la couleur du crayon : permet de choisir la couleur avec laquelle écrit la tortue à l'aide d'une palette de couleurs. Disponible également avec la primitive `fcc`.
- Options –> Choisir la couleur du fond : même chose avec le fond d'écran. Disponible avec la primitive `fcfg`.
- Options –> Définir les fichiers de démarrage : permet de définir des chemins vers des fichiers dit « de démarrage ». Toutes les procédures contenues dans ces fichiers au format *.lgo deviendront alors des « pseudo-primitives » du langage XLogo. Elles ne sont pas éditables ni modifiables

par l'utilisateur. Vous pouvez ainsi définir des primitives personnalisées. Vous pouvez de plus lui donner une commande (en logo) à effectuer au démarrage de XLogo. Vous avez ainsi la possibilité de lancer un programme que vous avez conçu dès l'ouverture de XLogo.

- Options –> Traduire des procédures : Ouvre une boîte de dialogue qui permet de traduire des commandes XLogo dans une langue désirée. (Très utile en particulier lorsqu'on récupère des sources Logo en anglais sur internet pour les remettre en français)
- Options –> Préférences : Ouvre une boîte de dialogue dans laquelle vous pouvez configurer plusieurs choses :
  - Langue : permet de choisir entre le français et l'anglais. Attention, les primitives changent d'une langue à l'autre.
  - Aspect : permet de définir le « look » de la fenêtre XLogo. Soit style Windows, style Java natif ou style Motif
  - Choisir la vitesse de défilement. Si vous souhaitez voir tous les déplacement de la tortue, vous pouvez la ralentir à l'aide de la barre prévu dans le premier onglet.
  - Dans le deuxième onglet, vous pouvez choisir votre tortue préférée.
  - Dans le troisième onglet, Plusieurs choses peuvent être fixées.
    → Tout d'abord, on peut fixer une taille limite à l'épaisseur du crayon. Si l'on ne veut pas utiliser cette limitation, mettre le nombre -1 dans la zone de texte associée.
    → Ensuite, on peut choisir la forme du crayon de la tortue, on ne se rend compte du choix de cette option que lorsque l'on choisit une épaisseur de crayon supérieure à 1.
    → On peut également fixer le nombre de tortues maximum en mode multitortues. (Par défaut 16)
    → On peut choisir d'effacer automatiquement la zone de dessin lorsqu'on sort de l'éditeur.
    → Vous pouvez choisir une taille personnalisée pour la zone de tracé. Par défaut XLogo se lance avec une zone de 1000 pixels sur 1000 pixels. Attention, lorsque vous agrandissez l'image, il peut-être nécessaire d'augmenter la taille mémoire attribuée à XLogo. Un message d'erreur vous en avertira.
    → Vous pouvez par conséquent également changer la valeur correspondant à l'espace mémoire alloué à XLogo. Par défaut, cette valeur est fixée à 64 Mo. Il se peut que vous soyez obligé de l'augmenter si vous souhaitez travailler sur une zone de dessin plus grande. Lorsqu'on modifie ce paramètre, le changement n'est effectif qu'après redémarrage de XLogo. Attention, n'augmentez pas abusivement sans raison cette valeur, cela peut considérablement ralentir votre système.
    → Enfin, Vous pouvez choir la précision du tracé. En haute qualité, vous n'aurez plus d'effet de crénelage en particulier. En revanche, bien penser qu'en augmentant la qualité, vous perdrez en rapidité d'exécution.
  - Dans le quatrième onglet, vous trouverez la liste des instruments que peut imiter votre carte son au travers de l'interface MIDI. Vous pouvez sélectionner un instrument précis en cliquant dessus. (Vous pouvez également également sélectionner un instrument avec la primitive `fixeinstrument numéro`. Malheureusement, il semble que la liste des instruments conserve un comportement un peu aléatoire (Matériel non détecté en particulier). J'espère que ce détail sera amélioré dans les versions futures. Voir la FAQ en fin d emanuel à ce sujet.
  - Dans le cinquième onglet, vous pouvez choisir la police de l'interface graphique ainsi que sa taille. Attention ceci n'affecte pas la police rendue par les primitives `ecris` et `etiquette`.

2.4 Menu « Aide »

- Menu –> Licence : Affiche la licence GPL sous laquelle est distribué ce logiciel.
- Menu –> Traduction : affiche une traduction de ladite licence. Cette traduction n'a aucune

valeur officielle, seule la version anglaise a ce rôle.

- Menu – > A propos : Classique .... et http ://xlogo.tuxfamily.org pour vos mises à jours ! ! o :)

Chapitre 3

Conventions adoptées dans XLOGO

Voici la présentation de certaines choses à savoir concernant le langage LOGO lui-même et d'autres concernant XLOGO spécifiquement.

3.1 Commandes et interprétation

Le langage LOGO permet de provoquer certains événements à l'aide de commandes internes : On appelle ces commandes les *primitives*. Chaque primitive attend un certain nombre de paramètres que l'on appelle *arguments*. Par exemple la primitive `ve` qui permet d'effacer l'écran ne prend aucun argument alors que la primitive `somme` attend deux arguments.

`somme 2 3` écrira 5 en retour.

Les arguments sont de trois types en LOGO :

- **Les nombres :** certaines primitives attendent des nombres comme argument : `av 100` est un exemple.
- **Les mots :** Les mots sont commencent tous par ". Un exemple de primitive pouvant travailler avec les mots est la primitive `ecris`. `ecris "bonjour` renvoie `bonjour`. A noter que si vous oubliez le ", l'interpréteur vous renverra un message d'erreur. En effet, `ecris` attend un argument, or pour l'interpréteur, `bonjour` ne représente rien puisque ce n'est ni un nombre, ni un mot, ni une liste ni une procédure déjà définie.
- **Les listes :** Elles sont définies entre crochets.

Les nombres sont traités soit en tant que valeur numérique, (Ex : `av 100`) mais également en tant que mot. (Ex : `ecris vide? 12` écris `faux`).

3.2 Procédures

En plus de ces primitives, vous pouvez définir vos propres commandes. On les appelle les *procédures*. Les procédures sont introduites à l'aide du mot `pour` et se terminent par le mot `fin`. On utilise l'éditeur de procédures interne à XLOGO pour les taper. Voici un petit exemple :

```
pour carre
repete 4[avance 100 tournedroite 90]
fin
```

Ces procédures ont le droit d'admettre également des arguments. Pour cela, on utilise des variables. Une variable est un mot auquel on peut affecter une valeur. voici un exemple très simple :

```
pour deux_fois :mot
ecris :mot
ecris :mot
fin

deux_fois [1 2 3] -----> 1 2 3
                         1 2 3
```

Voir tous les exemples de procédures en fin de manuel.

3.3 Le caractère spécial « \ »

Le caractère « \ » (backslash) permet en particulier de créer des mots contenant des espaces ou contenant un retour à la ligne. « \n » provoque un retour à la ligne et « \ » suivi d'un espace désigne un espace dans un mot.
Exemple :

```
ec "xlogo\ xlogo
xlogo xlogo
ec "xlogo\nxlogo
xlogo
xlogo
```

Il s'ensuit que l'on ne peut plus accéder au caractère « \ » en le tapant il faudra taper « \\ ».

De même, les caractères « ( ) [ ] # » sont des délimiteurs du langage Logo qui ne peuvent être utilisés dans des mots. On pourra les introduire en rajoutant un caractère « \ » devant.

Tout caractère « \ » tout seul est ignoré. Cette remarque est très importante en particulier pour la gestion des fichiers

Pour fixer le répertoire courant à `C :\Mes Documents`, il faudra taper :

```
fixerepertoire "c:\\Mes\ Documents.
```

Noter l'utilisation du « \ » pour notifier l'espace entre « Mes » et « Documents ». Si d'autre part, vous omettez le double backclash, le chemin défini sera alors `c :Mes Documents` et l'interpréteur rendra un message d'erreur.

3.4 Règles concernant les majuscules et minuscules

XLOGO ne fait pas la différence majuscule minuscule en ce qui concerne les noms des procédures et des primitives. Ainsi, avec la procédure `carre` définie précédemment, que vous tapiez `CARRE` ou `cArRe`, l'interpréteur de commande traduira correctement et exécutera `carre`. En revanche, XLOGO respecte les majuscules dans les listes et les mots :

```
ecris "Bonjour ----> "Bonjour (on garde le B majuscule)
```

3.5 Opérateurs et syntaxe

Il y a deux façons d'écrire certaines commandes. Par exemple, pour effectuer l'addition de 4 et de 7, il y a deux possibilités : soit on se sert de la primitive `somme` qui attend deux arguments : `somme 4 7` , soit on se sert de l'opérateur + : `4+7`. Les deux ont le même effet.
Voici la liste des correspondances entre opérateurs et primitives :

somme	difference	produit	divise
+	-	*	/
ou	et	egal ?	
| (ALT GR+6)	&	=	

Les deux opérateurs | et & sont deux opérateurs spécifiques à XLOGO. Ils n'existent pas dans les versions traditionnelles de LOGO. Voici quelques exemples d'utilisation :

```
ec 3+4=7-1 ----> vrai
ec 3=4 | 7=49/7 ----> vrai
ec 3=4 & 7=49/7 ----> faux
```

3.6 Les couleurs

Les couleurs sont définies dans XLogo à l'aide de trois nombres compris entre 0 et 255. Ce système de codage s'appelle le codage « RGB »(Red, Green, Blue). Chaque nombre correspond respectivement à l'intensité du rouge, du vert et du bleu dans la couleur considérée. Etant donné que ce codage n'est pas très intuitif, XLogo vous propose également 16 couleurs prédéfinies accessibles soit par un numéro soit par une primitive.

Numéro	Primitives	[R G B]	Couleur
0	`noir`	[0 0 0]	
1	`rouge`	[255 0 0]	
2	`vert`	[0 255 0]	
3	`jaune`	[255 255 0]	
4	`bleu`	[0 0 255]	
5	`magenta`	[255 0 255]	
6	`cyan`	[0 255 255]	
7	`blanc`	[255 255 255]	
8	`gris`	[128 128 128]	
9	`grisclair`	[192 192 192]	
10	`rougefonce`	[128 0 0]	
11	`vertfonce`	[0 128 0]	
12	`bleufonce`	[0 0 128]	
13	`orange`	[255 200 0]	
14	`rose`	[255 175 175]	
15	`violet`	[128 0 255]	
16	`marron`	[153 102 0]	

```
# Ces trois commandes ont le même effet.
fcc orange
fcc 13
fcc [255 200 0]
```

Chapitre 4

Liste des primitives

Comme il a été indiqué auparavant, le contrôle de la tortue s'effectue à l'aide de commandes internes appelées « primitives ». Voici une classification de ces primitives :

4.1 Déplacement de la tortue, gestion du crayon et des couleurs

Ce premier tableau regroupe les primitives qui permettent de déplacer la tortue.

Primitives	Arguments	Utilisation
`avance`, `av`	n : nombre de pas	Fait avancer de n pas la tortue suivant l'orientation courante.
`recule`, `re`	n : nombre de pas	Fait reculer de n pas la tortue suivant l'orientation courante.
`tournedroite`, `td`	n : angle	Fait tourner la tortue de n degrés vers la droite par rapport à son orientation actuelle.
`tournegauche`, `tg`	n :angle	Fait tourner la tortue de n degrés vers la gauche par rapport à son orientation actuelle.
`cercle`	R : nombre	Trace un cercle de rayon R autour de la tortue.
`arc`	R : nombre, cap1,cap2 :nombres	Trace un arc de cercle de rayon R autour de la tortue. Cette arc est compris entre les caps cap1 et cap2.
`origine`	aucun	Replace la tortue à sa position initiale, c'est à dire au point de coordonnées [0 0] et avec pour cap 0
`fixeposition`, `fpos`	[x y] : liste de deux nombres.	Déplace la tortue au point de coordonnées spécifié à l'aide de la liste des deux nombres.(x désigne l'abscisse et y l'ordonnée)
`fixex`	x : abscisse	Déplace la tortue horizontalement jusqu'au point d'abscisse x
`fixey`	y : ordonnée	Déplace la tortue verticalement jusqu'au point d'ordonnée y
`fixexy`	x y : abscisse puis ordonnée	Analogue à fpos[x y]
`fixecap`	n : cap	Oriente la tortue au cap spécifié. 0 correspond à la position verticale vers le haut. On tourne ensuite dans le sens des aiguilles d'une montre.
`etiquette`	a : mot ou liste	Dessine le mot ou la liste spécifiée à l'endroit où se trouve la tortue et suivant son inclinaison. Exemple : `etiquette [Salut à toi]` va écrire la phrase "Salut à toi" à l'endroit où est placé la tortue en respectant le cap de celle-ci.

Primitives	Arguments	Utilisation
`point`	a : liste	Le point défini par les coordonnées de la liste s'allume (dans la couleur du crayon).

Ce deuxième tableau regroupe les primitives permettant d'agir sur les propriétés de la tortue. Par exemple, faut-il que la tortue soit visible à l'écran ? De quelle couleur doit-elle écrire lorsqu'elle se déplace ?

Primitives	Arguments	Utilisation
`montretortue`, `mt`	aucun	Rend la tortue visible à l'écran.
`cachetortue`, `ct`	aucun	Rend la tortue invisible à l'écran.
`videecran`, `ve`	aucun	Efface la zone de dessin et réinitialise la tortue à sa position initiale.
`nettoie`	aucun	Efface la zone de dessin mais laisse la tortue au même endroit.
`baissecrayon`, `bc`	aucun	La tortue écrit lorsqu'elle se déplace.
`levecrayon`, `lc`	aucun	La tortue n'écrit pas lors d'un déplacement.
`gomme`, `go`	aucun	La tortue efface tous les traits qu'elle rencontre.
`inversecrayon`, `ic`	aucun	Abaisse le crayon et met la tortue en mode d'inversion.
`dessine`, `de`	aucun	Abaisse le crayon et le met en mode dessin classique.
`animation`	vrai ou faux	• Si l'argument vaut `vrai` : On passe en mode animation. La tortue ne dessine plus à l'écran mais effectue le tracé en mémoire. Pour actualiser le dessin à l'écran, utiliser la primitive `rafraichis`. Très utile pour créer une animation ou effectuer un tracé plus rapidement. • Si l'argument vaut `faux` : On repasse en mode classique. On voit les déplacements de la tortue à l'écran.
`rafraichis`	aucun	En mode animation, rafraichit l'écran : l'image sur la zone de dessin est actualisée.
`fixecouleurcrayon`, `fcc`	a : entier ou liste [r g b]	Fixe la couleur du crayon. Voir p.11.
`fixecouleurfond`, `fcfg`	a : entier ou liste [r g b]	Fixe la couleur du fond d'écran. Voir p.11.
`pos`	aucun	Retourne la position courante de la tortue. Ex : `pos` retourne [10 -100]
`cap`	aucun	Retourne le cap de la tortue (cf `fixecap`)
`vers`	a : liste	La liste doit contenir deux nombres représentant des coordonnées. Rend le cap qu'il faut donner à la tortue pour aller vers le point défini par les coordonnées de la liste.

Primitives	Arguments	Utilisation
`distance`	a : liste	La liste doit contenir deux nombres représentant des coordonnées. Rend le nombre de pas entre la position actuelle et le point défini par les coordonnées de la liste.
`couleurcrayon,cc`	a : liste	Retourne la couleur actuelle du crayon. Cette couleur est déterminée à l'aide d'une liste [r g b] ou r est la composante rouge, b la bleue et g la verte.
`couleurfond,cf`	a : liste	Retourne la couleur actuelle du fond. Cette couleur est déterminée à l'aide d'une liste [r g b] ou r est la composante rouge, b la bleue et g la verte.
`enroule,enr`	aucun	Si la tortue sort de la zone de dessin, elle réapparaît de l'autre côté !
`fen, fenetre`	aucun	La tortue est libre de sortir de la zone de dessin. Bien sûr, elle n'écrira pas en dehors de cette dernière.
`clos`	aucun	La tortue est confinée à la zone de dessin. Si elle s'apprête à sortir, un message d'erreur vous l'indiquera et vous donnera le nombre de pas maximum de la tortue avant sortie (à 1 ou 2 pas près ...).
`trouvecouleur,tc`	a : liste	Retourne la couleur du pixel de coordonnées a. Cette couleur est déterminée à l'aide d'une liste [r g b] ou r est la composante rouge, b la bleue et g la verte.
`fixetaillecrayon`, `ftc`	n : nombre	Définit l'épaisseur de la pointe du crayon en pixel. Réglé sur 1 par défaut.
`fforme`, `fixeforme`	n : nombre	Vous pouvez choisir de l'aspect de la tortue utilisée soit en allant dans Option-Préférences-Choix de la tortue soit à l'aide de cette primitive. Le nombre n doit être un entier compris entre 0 et 6. (0 désigne la forme triangulaire)
`forme`	aucun	Renvoie le numéro qui représente l'image actuelle de la tortue.
`fixetaillepolice`, `ftp`	n : entier	Lorsqu'on écrit du texte sur l'écran à l'aide de la primitive `etiquette`, il est possible de modifier la taille de la police utilisée à l'aide de cette primitive. Par défaut, la taille de la police est réglée à 12.
`taillepolice, tp`	aucun	Renvoie la taille de la police actuellement utilisée lorsqu'on écrit avec la primitive `etiquette`.
`fixenompolice`, `fnp`	n : entier	Fixe la police utilisée pour écrire à l'écran à l'aide de la primitive `etiquette`. Le numéro identifiant la police à utiliser est repérable dans Menu –> Options –> Préférences –> Onglet Police.

Primitives	Arguments	Utilisation
`nompolice`, `np`	aucun	Renvoie une liste composée de deux éléments. Le premier est le numéro correspondant à la police utilisée pour écrire à l'aide de la primitive `etiquette`. Le second est une liste contenant le nom de cette même police.
`fixeseparation`, `fsep`	a : nombre	Détermine le ratio entre la fenêtre graphique et la zone d'historique. Le nombre « `a` » doit être compris entre 0 et 1. Lorsqu'il vaut 1 la zone de dessin occupe toute la place, lorsqu'il vaut 0, la zone d'historique occupe toute la fenêtre etc
`separation`, `sep`	aucun	Renvoie le ratio actuel entre la zone de dessin et la zone d'historique.
`message, msg`	[liste]	Affiche un message d'information dans une boîte de dialogue, l'exécution du programme est stoppé en attente d'un click sur OK.

```
message [Ceci est un message d'information. Le texte est automatiquement disposé sur plusieurs lignes afin qu'il soit bien lisible. De plus, le programme reste en attente que l'on appuie sur OK.]
```

4.2 Affichage du texte dans la zone d'historique

Ce tableau regroupe les primitives associées à la zone de texte d'historique. Toutes les primitives concernant la taille et la couleur de la police utilisée ne sont valables que pour le rendu de la primitive `ecris`.

Primitives	Paramètres	Utilisation
`vt, videtexte`	aucun	Efface la zone contenant l'historique des commandes et des commentaires.

`ec, ecris`	mot, liste ou nombre	Affiche l'argument spécifié dans la zone d'historique.
		
		`ec "abcd --------> abcd`
		`ec [1 2 3 4] ----> 1 2 3 4`
		`ec 4 ------------> 4`

`tape`	mot, liste ou nombre	Identique à la primitive `ecris` mais ne retourne pas à la ligne.
`ftpt, fixetaillepolicetexte`	a : nombre	Définis la taille de la police dans la zone d'historique. Valable uniquement pour la primitive `ecris`
`taillepolicetexte, tpt`	aucun	Renvoie la taille de la police associée à la primitive `ecris`.
`fct, fixecouleurtexte`	a :nombre ou liste	Définis la couleur de la police dans la zone d'historique. Valable uniquement pour la primitive `ecris`. Voir p.11.
`ctexte, couleurtexte`	aucun	Renvoie la couleur de la police associée à la primitive `ecris` dans la zone d'historique.
`fixenompolicetexte, fnpt`	n : entier	Fixe la police utilisée pour écrire dans l'historique à l'aide de la primitive `ecris`. Le numéro de la police est repérable dans Menu –> Options –> Préférences –> Onglet Police.
`nompolicetexte, npt`	aucun	Renvoie une liste composée de deux éléments. Le premier élément est le numéro représentant la police utilisée pour écrire à l'écran à l'aide de la primitive `ecris`. Le second est une liste contenant le nom de cette même police.
`fixestyle, fsty`	liste ou mot	Fixe le style du rendu de la police utilisée par la primitive `ecris`. Les différents styles possibles sont `aucun, gras, italique, barre, indice, exposant, souligne`. Si vous souhaitez en utiliser plusieurs à la fois, les indiquer dans une liste. Voir les exemples après ce tableau.
`style, sty`	aucun	Renvoie une liste composée des différents styles actuellement utilisés pour le rendu de la primitive `ecris`.

Quelques exemples pour le formatage du texte avec la primitive `ecris` :

```
fixestyle [gras souligne] ecris "bonjour
```
bonjour
```
fsty "barre tape [texte rayé] fsty "italique tape "\ x fsty "exposant ecris 2
```
~~texte rayé~~ x^2

4.3 Opérations arithmétiques et logiques

Primitives	Arguments	Utilisation
`somme`	a b : nombres à additionner	Additionne les deux nombres a et b puis retourne le résultat. Ex : `somme` 40 60 retourne 100
`difference`	a b : nombres à soustraire	Retourne a-b. Ex : `difference` 100 20 retourne 80
`moins`	a : nombre	Retourne l'opposé de a. Ex : `moins` 5 retourne -5 .Voir la remarque à la suite de ce tableau
`produit`	a b : nombres	Retourne le produit de a par b.
`div, divise`	a b : nombres	Retourne le quotient de a par b `divise` 15 6 retourne 2.5
`quotient`	a b : nombres	Retourne le quotient entier de a par b `quotient` 15 6 retourne 2
`reste`	a b : entiers	Retourne le reste de la division de a par b.
`arrondi`	a : nombre	Retourne l'entier le plus proche du nombre a. `arrondi` 6.4 renvoie 6
`tronque`	a : nombre	tronque à l'unité le nombre a. `tronque 6.8` renvoie 6
`puissance`	a b : entiers	Renvoi a élevé à la puissance b. `puissance` 3 2 renvoie 9
`racine, rac`	n : nombre	renvoie la racine carrée de n
`log10`	n : nombre	renvoie le logarithme décimal de n
`sinus, sin`	a : nombre	renvoie le sinus du nombre a. (a est exprimé en degré)
`cosinus, cos`	a : nombre	Renvoie le cosinus du nombre a. (a est exprimé en degré)
`tan,tangente`	a : nombre	Renvoie le tangente du nombre a. (a est exprimé en degré)
`acos,arccosinus`	a : nombre	Renvoie l'angle dont le cosinus vaut a. (l'angle est exprimé en degré)
`asin,arcsinus`	a : nombre	Renvoie l'angle dont le sinus vaut a. (l'angle est exprimé en degré)
`atan,arctangente`	a : nombre	Renvoie l'angle dont la tangente vaut a. (l'angle est exprimé en degré)
`pi`	aucun	Renvoie le nombre π (3.141592653589793)
`hasard`	n : entier	Renvoie un nombre aléatoire compris entre 0 et n-1.
`absolue, abs`	n : nombre	Renvoie la valeur absolue (distance à zéro) du nombre proposé.

Remarque : Attention aux primitives nécessitant deux paramètres !

Ex : `fixexy` a b Si b est négatif
Par exemple, `fixexy` 200 -10

L'interpréteur logo va effectuer l'opération 200-10. Il va donc considérer qu'il n'y a qu'un paramètre (190) alors qu'il lui en faut deux d'où un message d'erreur. Pour éviter ce type de problème, utiliser la primitive "`moins`" indiquant l'opposé. `fixexy 200 moins 10` et là, il n'y a plus de problèmes !

Une autre possibilité peut être d'utiliser les parenthèses en tapant : `fixexy 200 (-10)`

Voici la liste des opérateurs logiques :

Primitives	Paramètres	Utilisation
`ou`	b : booléens	Renvoie vrai si a ou b est vrai, sinon renvoie faux
`et`	b : booléens	Renvoie vrai si a et b sont égaux à vrai sinon renvoie faux
`non`	a : booléens	Renvoie la négation de a. Si a est vrai, renvoie faux.Si a est faux, renvoie vrai.

4.4 Opérations sur les listes et les mots

Primitives	Paramètres	Utilisation
`mot`	a b	Concatène les deux mots a et b. Exemple : `ec mot "a 1` renvoie `a1`
`liste`	a b	Retourne une liste composée de a et b. Par exemple, `liste` 3 6 renvoie [3 6]. `liste` “une “liste renvoie [une liste]
`phrase`, `ph`	a b	Retourne une liste composée de a et b. Si a ou b est une liste, alors chacun des composants de a ou b devient élément de la liste créée (les crochets sont supprimés). Ex : `ph` [4 3] “bonjour renvoie [4 3 bonjour] `ph` [comment ça] “va renvoie [comment ça va]
`metspremier`, `mp`	a b : a quelconque, b liste	Insère a en première position de la liste b. Ex : `mp` “coucou [2] renvoie [coucou 2]
`metsdernier`, `md`	a b : a quelconque, b liste	Insère a en dernière position de la liste b Ex : `md` 5 [7 9 5] renvoie [7 9 5 5]
`inverse`	a : liste	Inverse l’ordre des éléments de la liste a `inverse` [1 2 3] renvoie [3 2 1]
`choix`	a : a mot ou liste	Si a est un mot, renvoie une des lettres de a au hasard.Si a est une liste, renvoie un des éléments de a au hasard.
`enleve`	a b : a quelconque, b liste	Enlève l’élément a de la liste b s’il apparaît dedans. Ex : `enleve` 2 [1 2 3 4 2 6] renvoie [1 3 4 6]
`item`	a b : a entier, b liste ou mot	Si b est un mot renvoie la lettre a du mot (1 désigne la première lettre). Si b est une liste renvoie l’élément numéro a de la liste.
`saufdernier`, `sd`	a : liste ou mot	Si a est une liste, renvoie toute la liste sauf le dernier élément. Si a est un mot, renvoie le mot sans sa dernière lettre.
`saufpremier`, `sp`	a : liste ou mot	Si a est une liste, renvoie toute la liste sauf le premier élément. Si a est un mot, renvoie le mot sans sa première lettre.
`dernier, der`	a : liste ou mot	Si a est une liste, renvoie le dernier élément de la liste. Si a est un mot, renvoie la dernière lettre du mot.
`premier, prem`	a : liste ou mot	Si a est une liste, renvoie le premier élément de la liste. Si a est un mot, renvoie la première lettre du mot.
`remplace`	li1 n li2 : li1 liste, n entier, li2 mot ou liste	Dans la liste `li`, remplace l’élément numéro `n` par le mot ou la liste proposé. `remplace [a b c] 2 8 --> [a 8 c]`
`ajoute`	li1 n li2 : li1 liste, n entier, li2 mot ou liste	Dans la liste `li`, ajoute en position numéro `n` le mot ou la liste proposé. `ajoute [a b c] 2 8 --> [a 8 b c]`

`compte`	a : liste ou mot	Si a est un mot, renvoie le nombre de lettres de a. Si a est une liste, renvoie le nombre d'éléments de a.
`unicode`	a : mot	Renvoie la valeur unicode du caractère « `a` ». `ec unicode "A` renvoie 65
`caractere, car`	a : nombre	Renvoie le caractère dont la valeur unicode est « `a` ». `ec caractere 65` renvoie `"A`

4.5 Booléens

Un booléen est une primitive qui renvoie le mot “vrai ou le mot “faux. Ces primitives se terminent par un point d'interrogation.

Primitives	Paramètres	Utilisation
`vrai`	aucun	Renvoie ”vrai
`faux`	aucun	Renvoie ”faux
`mot?`	a	Renvoie vrai si a est un mot, faux sinon.
`nombre?`	a	Renvoie vrai si a est un nombre, faux sinon.
`entier?`	a	Renvoie vrai si a est un entier, faux sinon.
`liste?`	a	Renvoie vrai si a est une liste, faux sinon.
`vide?`	a	Renvoie vrai si a est une liste vide ou un mot vide, faux sinon.
`egal?`	a b	Renvoie vrai si a et b sont égaux, faux sinon.
`precede?`	a b : mots	Renvoie vrai si a est avant b dans l'ordre alphabétique, faux sinon.
`membre?`	a b	Si b est une liste, précise si a est élément de b. Si b est un mot, précise si a est un caractère de b.
`membre`	a b	Si b est une liste, recherche a dans cette liste, deux cas possibles : -Si a est dans b, renvoie la sous-liste générée à partir de la première apparition de a dans b. -Si a n'est pas dans b, renvoie le mot faux. Si b est un mot, recherche le caractère a dans b, deux cas possibles : - Si a est dans b renvoie la fin du mot à partir de a. -Sinon, renvoie le mot faux. `membre` “o “coucou renvoie oucou `membre` 3 [1 2 3 4] renvoie [3 4]
`bc?`, `baissecrayon?`	aucun	Renvoie le mot vrai si le crayon est baissé, faux sinon.
`visible?`	aucun	Renvoie le mot vrai si la tortue est visible, faux sinon.
`prim?`, `primitive?`	a : mot	Renvoie vrai si le mot est une primitive de XLOGO, faux sinon.
`proc?`, `procedure?`	a : mot	Renvoie vrai si le mot est une procédure définie par l'utilisateur, faux sinon.

4.6 Effectuer un test à l'aide de la primitive `si`

Comme dans tout langage de programmation, le Logo offre la possibilité de vérifier si une condition donnée est vraie ou fausse afin d'exécuter le bout de code associé.

La primitive `si` permet de réaliser ces tests. Voici sa syntaxe :

```
si expression_test [ liste1 ] [ liste2 ]
```

Si l'expression test est vraie alors les commandes situées dans la liste 1 seront exécutées. En revanche, si cette expression est fausse alors ce sont les commandes de la liste 2 qui seront exécutées. La deuxième liste d'intruction est optionnelle.
Exemples d'utilisation :

- `si 1+2=3[ecris "vrai][ecris "faux]`
- `si (premier "XLOGO)="Y [av 100 td 90] [ec [ XLOGO commence par un X !] ]`
- `si (3*4)=6+6 [ec 12]`

4.7 Gestion des procédures et des variables

4.7.1 Les procédures

Les procédures sont des sortes de « programmes ». A l'appel de leur nom, les instructions comprises dans le corps de la procédure sont exécutées. On définit une procédure à l'aide du mot-clé **pour**.

```
pour nom_de_la_procédure :v1 :v2 :v3 ....
Corps de la procédure
fin
```

nom_de_la_procédure est le nom donnée à la procédure.
:v1 :v2 :v3 représentent les variables utilisées au sein de cette procédure (variables locales).
Corps de la procédure représente les instructions à exécuter à l'appel de cette procédure.

Ex :

```
pour carre :c
repete 4[av :c td 90]
fin
```

La procédure se nomme `carre` et possède un paramètre s'appelant c. `carre` 100 produira donc un carre de côté 100. (Voir les exemples de procédures à la fin du manuel.)

Il est possible depuis la version 0.7c de rajouter des commentaires dans le code en les précédant du signe #.

```
pour carre :c
#cette procédure permet de tracer un carré de côté donné :c.
repete 4[av :c td 90] # pratique, non?
fin
```

IMPORTANT : Il ne doit pas y avoir de commentaires sur la ligne du `pour` et sur celle de `fin`.

4.7.2 Notion de variables

Il existe deux sortes de variables :

- Les variables globales : elles sont toujours disponibles à n'importe quel endroit du programme.
- Les variables locales : elles ne sont accessibles que dans la procédure où elles ont été définies.

Dans cette version de LOGO, les variables locales ne sont pas accessibles dans les sous-procédures. A la sortie de la procédure, les variables locales sont éliminées.

4.7.3 La primitive 'trace'

Il est possible pour suivre le déroulement d'un programme de lui faire afficher les procédures en cours d'exécution. Ce mode permet d'afficher également si les procédures rendent des arguments à l'aide de la primitive `retourne`. Pour enclencher ce mode, on tape :

```
trace vrai
```

Bien sûr, `trace faux` désactivera le mode « trace ». Un petit exemple avec le factorielle (voir p. 43).

```
trace vrai ecris fac 4
fac 4
  fac 3
    fac 2
      fac 1
      fac retourne 1
    fac retourne 2
  fac retourne 6
fac retourne 24
24
```

Primitives	Paramètres	Utilisation
`donne`	a b : a mot, b quelconque	Si la variable locale a existe, lui affecte la valeur b. Sinon, crée la variable globale a en lui affectant la valeur b. Ex : `donne` “a 100 affecte 100 à la variable a
`locale`	a : mot	Crée une variable nommée a. Attention, celle-ci n'est pas initialisée. Pour lui donner une valeur, voir `donne`.
`donnelocale`	a b : a mot, b quelconque	Crée une nouvelle variable locale a et lui affecte la valeur b.

`def, definis`	mot1 liste2 liste3	Définis une nouvelle procédure nommée mot1, munie des variables contenues dans liste2 et dont les instructions à exécuter sont contenues dans liste3.
		
		`def "polygone [nb longueur]`
		`[repete :nb[av :longueur td 360/:nb]]`
		
		—-> Ceci définit une procédure nommée polygone avec deux variables (`:nb` et `:longueur`). Elle permet de tracer un polygône régulier dont on peut choisir le nombre de côtés et la longueur de chacun des côtés.

`chose`	a : mot	Renvoie la valeur de la variable a. chose ”a et :a sont deux notations équivalentes.
`efn, effacenom`	a : mot	Efface la procédure s'appelant a.
`efv, effacevariable`	a : mot	Efface la variable a.
`efns, effacenoms`	aucun	Efface toutes les variables et procédures en cours.
`imts`	aucun	Enumère toutes les procédures actuellement définies.
`listevariables`	aucun	Renvoie une liste contenant l'ensemble des variables actuellement définies.
`exec, execute`	a :liste	Exécute la liste d'instruction contenue dans la liste.

4.8 Gestion des Fichiers

Primitives	Arguments	Utilisation
`chargeimage, ci`	a : mot	Affiche le fichier image contenu dans la liste. Son coin supérieur gauche sera placé où se trouve la tortue . Les formats supportés sont le png et le jpg. Le chemin spécifié doit être relatif par rapport au répertoire courant. Ex : `chargeimage "tortue.jpg]`
`cat, catalogue`	aucun	Liste le contenu du répertoire par défaut. (Equivalent de la commande `ls` pour linux et `dir` pour DOS)
`frep, fixerepertoire`	m : mot	Fixe le répertoire en cours. Le chemin doit être absolu et doit être spécifié à l'aide d'un mot.
`changedossier, cd`	m : mot	Permet de choisir le répertoire courant. Le chemin est relatif par rapport au répertoire courant actuel. On peut utiliser la notation « `..` » pour faire référence au répertoire parent.
`rep, repertoire`	aucun	Rend le répertoire en cours. Par défaut, il est fixé au répertoire utilisateur c'est à dire `/home/votre_login` pour les linuxiens, `C :\WINDOWS` pour les autres.
`sauve`	m : mot l :liste	Un bon exemple pour expliquer cela : `sauve "essai.lgo [proc1 proc2 proc3]` sauve dans le fichier essai.lgo du répertoire courant les procédures proc1, proc2 et proc3. Si l'extension .lgo est omise, elle est rajoutée par défaut. Le mot spécifié désigne un chemin relatif par rapport au répertoire courant. Cette commande ne fonctionne pas avec un chemin absolu.
`sauved`	m : mot	`sauved "test.lgo` sauve dans le fichier test.lgo du répertoire courant l'ensemble des procédures définies actuellement. Si l'extension .lgo est omise, elle est rajoutée par défaut. Le mot spécifié désigne un chemin relatif par rapport au répertoire courant. Cette commande ne fonctionne pas avec un chemin absolu.
`ramene`	m : mot	Ouvre et interprete le fichier m. Par exemple, pour effacer toutes les procédures définies et charger le fichier essai.lgo, on écrira : `efns charge "essai.lgo`. Le mot spécifié désigne un chemin relatif par rapport au répertoire courant. Cette commande ne fonctionne pas avec un chemin absolu.
`ouvreflux`	`id fich`	Lorque l'on veut lire ou écrire dans un fichier, il faut au préalable ouvrir un flux vers ce fichier. L'argument `fich` doit être le nom du fichier considéré. On doit utiliser un mot pour indiquer le le nom du fichier dans le répertoire courant. L'argument `id` est le numéro que l'on affecte à ce flux afin de pouvoir l'identifier.

Primitives	Arguments	Utilisation
`listeflux`	aucun	Affiche la liste des différents flux ouverts avec leur identifiant.
`lisligneflux`	`id`	Ouvre le flux dont l'identifiant est le numéro passé en argument puis lis une ligne dans ce fichier.
`liscarflux`	`id`	Ouvre le flux dont le numéro d'identifiant est celui passé en argument puis lis un caractère dans ce fichier. Cette primitive renvoie un nombre représentant la valeur du caractère (semblable à `liscar`).
`ecrisligneflux`	`id liste`	Ecris la ligne de texte contenue dans la liste au début du fichier repéré par l'identifiant `id`. Attention, l'écriture n'est effective que lorsque l'on ferme le flux avec la primitive `fermeflux`.
`ajouteligneflux`	`id liste`	Ecris la ligne de texte contenue dans la liste à la fin du fichier repéré par l'identifiant `id`. Attention, l'écriture n'est effective que lorsque l'on ferme le flux avec la primitive `fermeflux`.
`fermeflux`	`id`	Ferme le flux dont le numéro d'identifiant est celui passé en argument.
`finflux ?`	`id`	Renvoie `"vrai` si on est arrivé à la fin du fichier. Renvoie `"faux` sinon.

Voici un exemple d'utilisation des primitives permettant de lire et écrire dans un fichier. Nous présenterons cet exemle pour une architecture de type Windows. Les autres utilisateurs sauront adapter l'exemple suivant.

L'objectif est de créer le fichier `c :\exemple` contenant les trois lignes :

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789

```
# On ouvre un flux vers le fichier désiré. Ce flux sera repéré par le numéro 2
fixerepertoire "c:\\
ouvreflux 2 "exemple
# On écrit les lignes désirées
ecrisligneflux 2 [ABCDEFGHIJKLMNOPQRSTUVWXYZ]
ecrisligneflux 2 [abcdefghijklmnopqrstuvwxyz]
ecrisligneflux 2 [0123456789]
# On ferme le flux pour achever l'écriture
fermeflux 2
```

A présent, on peut constater que l'écriture s'est bien passée :

```
# On ouvre un flux vers le fichier à lire. Ce flux sera repéré par le numéro 0
ouvreflux 0 "c:\\exemple
# On lit les lignes du fichiers successivement
ec lisligneflux 0
ec lisligneflux 0
ec lisligneflux 0
# On ferme le flux
```

```
fermeflux 0
```

Si on souhaite à présent rajouter la ligne « Formidable ! » :

```
fixerepertoire "C:\\
ouvreflux 1 "exemple]
ajouteligneflux 1 [Formidable !]
fermeflux 1
```

4.9 Fonction avancée de remplissage :

Il existe deux primitives permettant de colorier une forme : La primitive `remplis` et la primitive `rempliszone` . On peut apparenter ces primitives avec la fonction "pot de peinture" utilisée dans de nombreux logiciels de retouche d'images. On peut atteindre les bords de la zone de dessin. Il y a deux règles à respecter pour utiliser correctement ces primitives :

1. Le crayon doit être en position baissé (`bc`).
2. La tortue ne doit pas être située sur un pixel de la couleur dont on veut remplir la forme. (Si on veut colorier en rouge, ne pas se trouver soi-même sur du rouge...)

Voyons un exemple pour expliquer la différence entre `remplis` et `rempliszone` : Le pixel sous la tortue

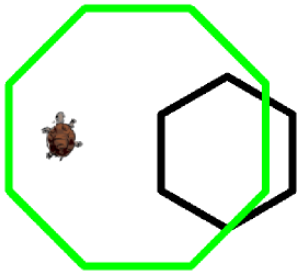


FIG. 4.1 – Situation initiale

est actuellement de couleur blanche. La primitive `remplis` va colorier tous les pixels blancs voisins avec la couleur du crayon en cours. Si par exemple on tape : `fcc 1 remplis`

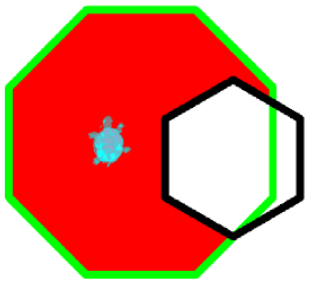


FIG. 4.2 – Avec la primitive `remplis`

Revenons à présent au premier cas, Si la couleur du crayon de la tortue est le noir, la primitive `rempliszone`, colorie tous les pixels voisins jusqu'à rencontrer la couleur en cours (ici noire). Voici, un bel exemple d'utilisation de la primtive `remplis` :

```
pour demice :c
# trace un demi-cercle de diamètre :c
repete 180 [av  :c*tan 0.5 td 1]
av :c*tan 0.5
td 90 av :c
fin

pour arcenciel :c
```

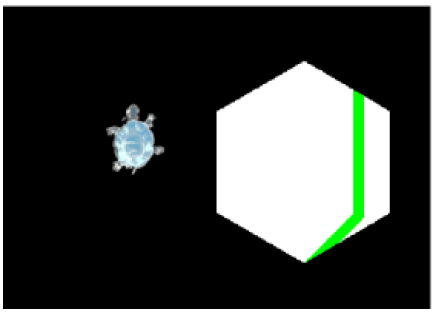


FIG. 4.3 – Avec la primitive `rempliszone`, en tapant : `fcc 0 rempliszone`

```
si :c<100 [stop]
demice :c td 180 av 20 tg 90
arcenciel :c-40
fin

pour dep
lc td 90 av 20 tg 90 bc
fin

pour demarrer
ct arcenciel 400 go tg 90 av 20 re 120 de lc td 90 av 20 bc
fcc 0 remplis dep
fcc 1 remplis dep
fcc 2 remplis dep
fcc 3 remplis dep
fcc 4 remplis dep
fcc 5 remplis dep
fcc 6 remplis dep
fin
```

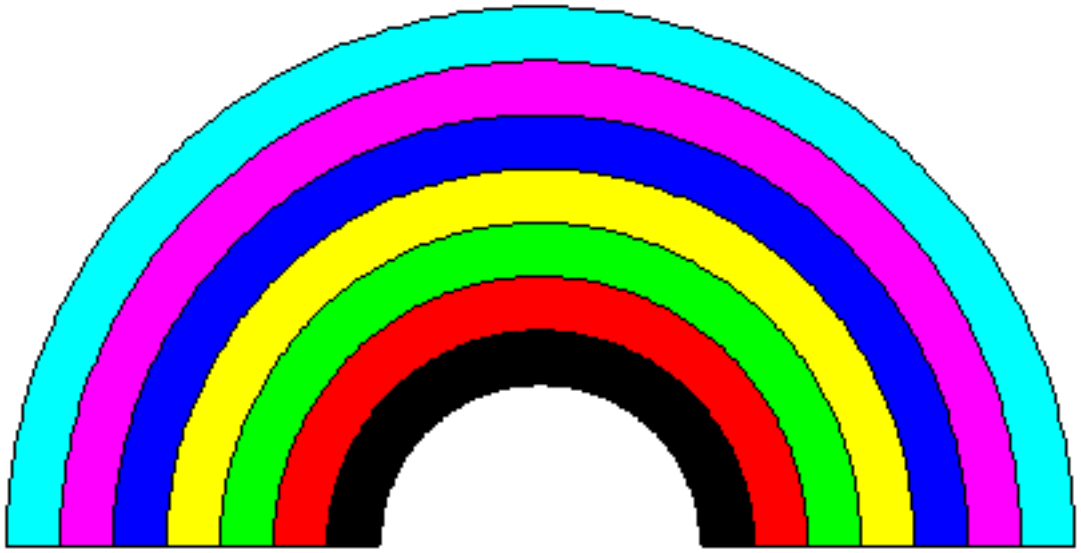


FIG. 4.4 – Arc-en-LOGO

4.10 Les instructions de saut

XLOGO possède trois instructions de saut : `stop`, `stoptout` et `retourne`.

- `stop` peut avoir deux effets : s'il est inclus dans une boucle `repete` ou `tantque` : à ce moment là, on sort de la boucle. S'il est dans une procédure, on sort immédiatement de la procédure.
- `stoptout` interrompt définitivement l'exécution de toutes les procédures en cours.
- `retourne` permet de sortir d'une procédure en retournant une valeur.

Voir les nombreux cas d'utilisation des deux primitives `stop` et `retourne` dans les exemples à la fin du manuel.

4.11 Le mode multitortues

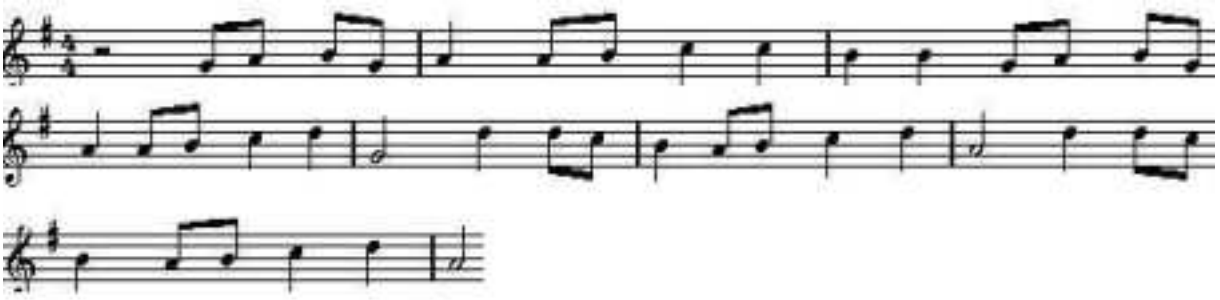
Il est possible de piloter à l'écran plusieurs tortues à la fois. Par défaut, lorsqu'on lance XLOGO, une seule tortue est active. Elle est repérée par le numéro 0. Pour "créer" une nouvelle tortue à l'écran, on utilise la primitive `fixetortue` suivi du numéro de la tortue désirée. Pour évitement l'encombrement sur la fenêtre de dessin, celle-ci est créée à l'origine de la zone de dessin (coordonnées (0 ;0)) et est invisible, c'est à dire qu'il faudra se servir de la commande `mt` pour la faire apparaître. Ensuite cette nouvelle tortue obéit aux ordres classiques tant que l'on ne change pas de tortue avec `fixetortue`. Le nombre maximal de tortues disponibles se règles dans Options - Préférences - Onglet options.

Voici la liste des primitives concernant le mode multitortues :

Primitives	Paramètres	Utilisation
`ftortue, fixetortue`	a : nombre	Détermine le numéro de la tortue active. Par défaut, la première tortue active au lancement de XLOGO a le numéro 0.
`tortue`	aucun	Renvoie le numéro de la tortue actuellement utilisée
`tortues`	aucun	Renvoie une liste composée de tous les numéros des tortues actuellement utilisées
`tuetortue`	a : nombre	Elimine de l'écran la tortue numéro a

4.12 Jouer de la musique

Primitives	Paramètres	Utilisation
`seq, sequence`	a : liste	Met en mémoire la séquence musicale située dans la liste. Pour apprendre à rédiger séquence musicale, voir les instructions après le tableau.
`joue`	aucun	Joue la séquence actuellement mise en mémoire.
`instr, instrument`	aucun	Renvoie le numéro correspondant à l'instrument actuellement sélectionné.
`finstr, fixeinstrument`	a : nombre	Sélectionne comme instrument l'instrument numéro a. Vous pouvez voir la liste des instruments disponibles dans menu-options-préférence-onglet son.
`indseq, indexsequence`	aucun	Renvoie à quel temps le curseur est situé dans la séquence en cours.
`findseq, fixeindexsequence`	a : nombre	Déplace le curseur au temps numéro a dans la séquence musicale actuellement en mémooire.
`efseq, effacesequence`	aucun	Efface la séquence actuellement en mémoire.

Pour jouer de la musique, il faut mettre au préalable la composition désirée en mémoire dans ce que l'on appellera ici une séquence musicale. On crée la séquence à l'aide de la commande `seq` ou `sequence`. Voici les quelques règles à respecter pour écrire convenablement une séquence musicale :
`do re mi fa sol la si` : désignent les notes usuelles de la première octave.
Pour faire un ré dièse, on tapera `re +`
Pour faire un re bémol, on tapera `re -`
Si on veut changer d'octave, on utilise le symbole " :" suivit de + ou -. Par exemple, après avoir tapé :++, toutes les notes jouées seront augmentées de deux octaves (il y a deux "+").
Les notes sont par défaut jouées sur une durée de un temps. Si on veut changer la durée d'une série de notes, on l'indique par le nombre indiquant la durée désirée. Pour taper une série de croches($\frac{1}{2}$ temps), on tapera `seq [0.5 sol la si]`.
Un bon exemple valant mieux que mille explications :

```
pour tabac
# Met en mémoire la partition
sequence [0.5 sol la si sol 1 la 0.5 la si 1 :+ do do :- si si 0.5 sol la si sol
          1 la 0.5 la si 1 :+ do re 2 :- sol ]
sequence [:+ 1 re 0.5 re do 1 :- si 0.5 la si 1 :+ do re 2 :- la ]
sequence [:+ 1 re 0.5 re do 1 :- si 0.5 la si 1 :+ do re 2 :- la ]
sequence [0.5 sol la si sol 1 la 0.5 la si 1 :+ do do :- si si 0.5 sol la si sol
          1 la 0.5 la si 1 :+ do re 2 :- sol ]
fin
```

Pour lancer la musique, il ne nous reste plus qu'à taper : `tabac joue`
Voyons à présent une application intéressante de la primitive findseq. Taper les commandes suivantes :

```
efseq       # On efface la séquence actuellement en mémoire
tabac       # On recharge la musique précédente
findseq 2   # On replace le curseur au niveau du premier "la" noir de la 2nde mesure
tabac       # On recharge la même séquence mais décalée de deux temps.
joue        # Un magnifique canon !
```

Vous pouvez également changer d'instruments, soit à l'aide de la commande `finstr` soit dans le menu Options-Préférences-Onglet son. Vous trouverez la liste de tous les instruments disponibles avec leur numéro (C'est en anglais, mais ça permet de se donner une idée. Chez moi, 411 instruments disponibles !)

4.13 Boucles :

XLOGO dispose de trois primitives permettant d'effectuer des boucles : `repete`, `repetepour` et `tantque`.

4.13.1 Une boucle avec `repete`

Voici la syntaxe de `repete` :

```
repete n liste_d_instruction
```

`n` est un entier et `liste_d_instruction` est une liste contenant des instruction à exécuter. L'interpréteur LOGO va effectuer n fois les commandes contenues dans la liste : cela évite ainsi de recopier n fois la même instruction !
Ex :

```
repete 4 [avance 100 tournegauche 90]      # Un carré de côté 100
repete 6 [avance 100 tournegauche 60]      # Un hexagone de côté 100
repete 360 [avance 2 tournegauche 1]       # Un euh... 360-gone de côté 2
                                           # Bref, quasiment un cercle !
```

Au sein d'une boucle `repete`, est définie une variable interne `compteur`. Celle-ci désigne le numéro de l'itération en cours (la première itération étant numérotée 1).

```
repete 3 [ec compteur]
1
2
3
```

4.13.2 Une boucle avec `repetepour`

`repetepour` joue le rôle des boucles `for` dans d'autres langages de programmation. Elle consiste à affecter à une variable un certain nombre de valeurs comprises dans un intervalle donné suivant un incrément donné. Voici sa syntaxe :

```
repetepour liste1 liste2
```

La liste 1 contient trois paramètres : le nom de la variable, la borne initiale, la borne finale.
On peut rajouter un quatrième argument optionnel désignant l'incrément (le pas avec lequel la variable défile). S'il est omis, il est par défaut de 1. Quelques exemples d'utilisation :

```
repetepour [i 1 4][ec :i*2]
2
4
6
8

# A présent on fait varier i entre 7 et 2 en descendant de 1.5 à chaque fois
# noter l'incrément négatif
# On affiche ensuite i son carré.

repetepour [i 7 2 -1.5 ][ec liste :i puissance :i 2]
```

```
7 49
5.5 30.25
4 16
2.5 6.25
```

4.13.3 Une boucle avec `tantque`

Voici la syntaxe de `tantque` :

```
tantque liste_a_tester liste_d_instruction
```

`liste_a_tester` est une liste contenant une suite d'instruction rendant un booléen. `liste_d_instruction` est une liste contenant des instruction à exécuter. L'interpréteur LOGO exécutera continuellement `liste_d_instruction` tant que `liste_a_tester` rend vrai.
Ex :

```
tantque ["vrai] [td 1]                    # La tortue tourne sur elle-même

# Un exemple qui nous permet d'épeler l'alphabet à l'envers

donne "liste "abcdefghijklmnopqrstuvwxyz
tantque [non vide? :liste] [ec dernier :liste donne "liste saufdernier :liste]
```

4.14 Intercepter des actions de l'utilisateur

XLOGO peut interagir avec l'utilisateur pendant l'exécution d'un programme par l'intermédiaire du clavier et de la souris.

4.14.1 Interaction avec le clavier

On peut donc recevoir du texte de l'utilisateur pendant le programme à l'aide de 3 primitives : `touche ?`, `liscar` et `lis`.
`touche ?` : rend vrai ou faux selon qu'une touche ait été pressée ou non depuis le début de l'exécution du programme.
`liscar` :

- Si `touche ?` est faux, bloque le programme jusqu'à ce l'utilisateur appuie sur une touche.
- Si `touche ?` est vrai rend la valeur correspondant à la touche qui a été la dernière enfoncée.

A —> 65	B —> 66	C —> 67	etc ...	Z —> 90
← —> -37 ou -226 (NumPad)	↑ —> -38 ou -224	→ —> -39 ou -227	↓ —> -40 ou -225	
Echap —> 27	F1 —> -112	F2 —> -113	....	F12 —> -123
Shift —> -16	Espace —> 32	Ctrl —> -17	Enter —> 10	

Tab. 4.1 – Quelques valeurs de touche

Si vous avez un doute par le mot retourné par une touche, il vous suffit de taper :
`ec liscar`. L'interpréteur va alors attendre que vous tapiez sur une touche puis vous donnera la valeur correspondante.

`lis liste_titre mot` : Affiche une boîte de dialogue dont le titre est `liste_titre`. L'utilisateur peut alors rentrer une réponse dans un champ de texte, la réponse sera stockée sous forme de liste dans la variable :mot lorqu'il validera ou cliquera sur le bouton OK.

4.14.2 Quelques exemples d'utilisation :

```
pour yeuv
lis [Quelle est ton age?] "age
donne "age premier :age
si :age<18 [ec [tu es mineur]]
si ou :age=18 :age>18 [ec [tu es majeur]]
si :age>99 [ec [Respect !!]]
fin
pour rallye
si touche? [
donne "car liscar
si :car=-37 [tg 90]
si :car=-39 [td 90]
si :car=-38 [av 10]
si :car=-40 [re 10]
si :car=27 [stop]
]
rallye
fin
# On contrôle la tortue avec le clavier, on arrête avec Esc
```

4.14.3 Intercepter certains événements provenant de la souris

Pour cela, on dispose de trois primitives : `lissouris`, `souris ?` et `possouris`.

`lissouris` : Bloque le programme jusqu'à ce qu'un événement souris se produise : on entend par événement souris le fait de déplacer la souris ou de cliquer sur l'un de ses boutons. Une fois l'événement produit, `lissouris` rend un nombre permettant de caractériser l'événement. Voici les différents codes associés aux différents événements qu'ils représentent :

0 –> on a déplacé la souris
1 –> on a appuyé sur le bouton 1 de la souris
2 –> on a appuyé sur le bouton 2 de la souris
etc
Les boutons sont numérotés de la gauche vers la droite (en principe...)

`possouris` : Renvoie une liste contenant les coordonnées de la souris lors du dernier événement intercepté.

`souris ?` : rend vrai ou faux selon que l'on ait agi ou non sur la souris depuis le début de l'exécution du programme.

4.14.4 Quelques exemples d'utilisation

Dans cette première procédure, la tortue suit la souris lorsqu'elle se déplace sur la zone de dessin.

```
pour exemple
# Si on déplace la souris, se positionner à la nouvelle position
si lissouris=0 [fpos possouris]
exemple
fin
```

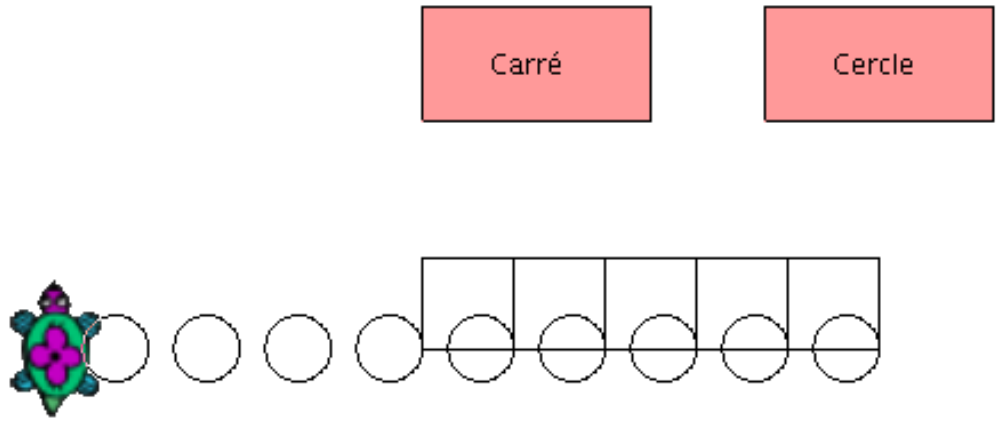
Dans cette deuxième procédure, c'est le même principe sauf qu'il faut cliquer avec le bouton gauche de la souris pour provoquer le déplacement de la tortue sur la zone de dessin.

```
pour exemple2
si lissouris=1 [fpos possouris]
exemple2
fin
```

Dans ce troisième exemple, nous allons créer deux boutons. Celui de gauche permettra de tracer un carré de 40 sur 40 vers la droite, celui de droite un petit cercle vers la gauche. Enfin, si l'on clique avec le troisième bouton de la souris sur le bouton de droite, cela provoquera l'arrêt du programme.

```
pour bouton
#crée un bouton rectangulaire de 50 sur 100 colorié en saumon
repete 2[av 50 td 90 av 100 td 90]
td 45 lc av 10 bc fcc [255 153 153]
remplis re 10 tg 45 bc fcc 0
fin

pour lance
```

```
ve bouton lc fpos [150 0] bc bouton
lc fpos [30 20] bc etiquette "Carré
lc fpos [180 20] bc etiquette "Cercle
lc fpos [0 -100] bc
souris
fin

pour souris
# On enregistre le résultat de lissouris dans la variable ev
donne "ev lissouris
# On enregistre la première coordonnée de la souris dans la variable x
donne "x item 1 possouris
# On enregistre la première coordonnée de la souris dans la variable y
donne "y item 2 possouris
# Si l'on clique sur le bouton de gauche
si :ev=1 & :x>0 & :x<100 & :y>0 & :y<50 [carre]
# Si l'on clique sur le bouton de droite
si  :x>150 & :x<250 & :y>0 & :y<50 [
          si :ev=1 [cercle]
          si :ev=3 [stop]
]
souris
fin

pour cercle
repete 90 [av 1 tg 4] tg 90 lc  av 40 td 90 bc
fin

pour carre
repete 4 [av 40 td 90] td 90 av 40 tg 90
fin
```



4.15 Gestion du temps

XLogo dispose de plusieurs primitives permettant de connaître l'heure, la date ou encore de gérer des comptes à rebours (utiles pour répéter une tâche à des intervalles fixés).

Primitives	Arguments	Utilisation
`attends`	n : entier	Bloque le programme et donc la tortue pendant n $60^{\text{ème}}$ de secondes.
`debuttemps`	n : entier	Lance un compte à rebours de n secondes. On peut savoir si le compte à rebours est terminé à l'aide de la primitive `fintemps ?`
`fintemps ?`	aucun	Rend `"vrai` si aucun compte à rebours n'est actif. Rend `"faux` si le compte à rebours n'est pas terminé.
`date`	aucun	Renvoie une liste formé de trois entiers représentant la date. Le premier indique le jour, le second le mois et le dernier l'année. —> [jour mois année]
`heure`	aucun	Renvoie une liste de trois entiers représentant l'heure. Le premier entier représente les heures, le second les minutes et le dernier les secondes. —> [heure minute seconde]
`temps`	aucun	Renvoie le temps écoulé depuis le démarrage de XLogo. Ce temps est exprimé en secondes.

Voici une petite procédure exemple :

```
pour horloge
# affiche l'heure sous forme numérique
# (on actualise l'affichage toutes les 5 secondes)
si fintemps? [
ve
fixepolice 75 ct
donne "heu heure
donne "h premier :heu
donne "m item 2 :heu
#affichage à deux chiffres des minutes (on rajoute le 0)
si :m-10<0 [donne "m mot 0 :m]
donne "s dernier :heu
#affichage à deux chiffres des secondes
si :s-10<0 [donne "s mot 0 :s]
etiquette mot mot mot mot :h ": :m ": :s
debuttemps 5
]
horloge
fin
```

4.16 Se servir du réseau avec XLogo

4.16.1 Le réseau : comment ça marche ?

Tout d'abord, dans cette introduction, il est nécessaire de vous expliquer certains termes de vocabulaire afin de bien comprendre l'usage des différentes primitives. Deux ordinateurs peuvent commuiquer

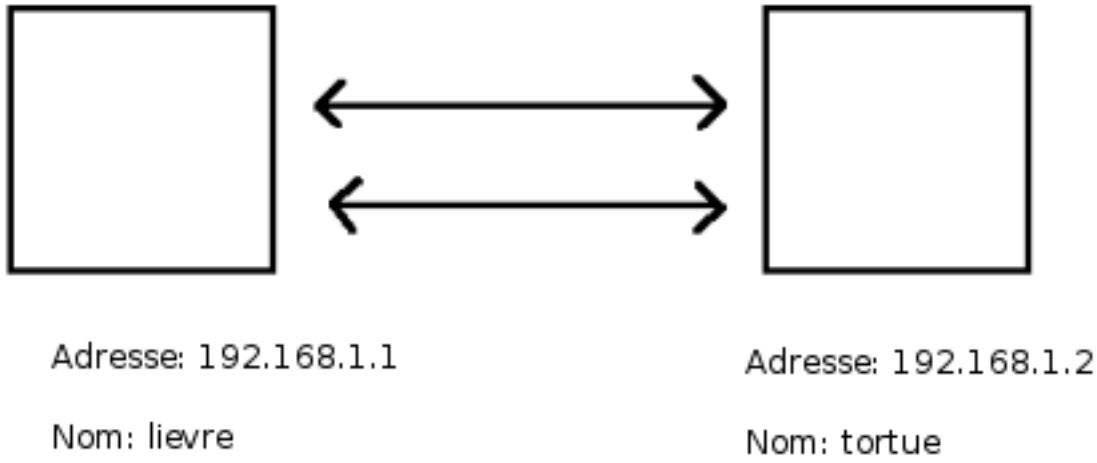


Fig. 4.5 – Notion de réseau

via le réseau s'ils sont équipés de carte réseau (appelée aussi carte ethernet). Chaque ordinateur est alors repéré par une adresse personnelle : *Son adressse IP*. Cette adresse IP est composée de 4 entiers compris entre 0 et 255 séparés par des points. Par exemple, l'adresse IP du premier ordinateur du schéma précédent est 192.168.1.1

Etant donné qu'il n'est pas facile de retenir ce genre d'adresse, il est également possible de faire correspondre à chaque adresse IP un nom usuel plus facile à retenir. Sur le schéma précédent, on peut ainsi s'adresser à l'ordinateur de droite soit en l'appelant par son adresse IP : 192.168.1.2, soit en l'appelant par son nom : `tortue`

Je ne m'étends pas davantage sur la signification de ces nombres. Je rajoute juste une chose qu'il est bon de savoir, l'ordinateur local sur lequel on travaille est repéré également par une adresse : 127.0.0.1. Le nom qui lui est associé est généralement `localhost`

4.16.2 Primitives orientées réesau

XLogo dispose de 4 primitives permettant de communiquer grâce au réseau : `ecoutetcp`, `executetcp`, `chattcp` et `envoietcp`. On prendra toujours dans les exemples qui suivent le cas des deux ordinateurs du schéma précédent.

- `ecoutetcp` : Cette primitive `ecoutetcp` est la base de toute communication réseau. Elle n'attend aucun argument. Elle permet de mettre l'ordinateur qui l'exécute à l'écoute d'ordres donnés par d'autres ordinateurs du réseau.

- `executetcp` : Cette primitive permet d'exécuter des instructions sur un ordinateur du réseau. Syntaxe : `executetcp mot liste` $\longrightarrow$ Le mot désigne l'adresse IP ou le nom de l'ordinateur appelé, la liste contient les instructions à exécuter.

Exemple : Je suis sur l'ordinateur `lievre`, je souhaite tracer un carré de côté 100 sur l'autre ordinateur. Par conséquent, il faut que sur l'ordinateur `tortue`, je lance la commande `ecoutetcp`. Ensuite, sur l'ordinateur `lievre`, je lance :

```
executetcp "192.168.1.2 [repete 4[av 100 td 90]]
ou
executetcp "tortue [repete 4[av 100 td 90]]
```

- `chattcp` : Permet de dialoguer entre deux ordinateurs du réseau en affichant une fenêtre permettant la conversation.
  Syntaxe : `chattcp mot liste` $\longrightarrow$ Le mot désigne l'adresse IP ou le nom de l'ordinateur appelé, la liste contient la phrase à afficher.

  Exemple : `lievre` veut discuter avec `tortue`.
  `tortue` lance `ecoutetcp` pour se mettre en attente de requête d'ordinateurs du réseau. `lievre` lance alors : `chattcp "192.168.1.2 [bonjour]`.
  Deux fenêtres permettant le dialogue s'ouvre alors sur chacun des ordinateurs.

- `envoietcp` : Envoie des données vers un ordinateur du réseau puis renvoie la réponse de l'autre ordinateur.
  Syntaxe : `envoietcp mot liste` $\longrightarrow$ Le mot désigne l'adresse IP ou le nom de l'ordinateur appelé, la liste contient les données à envoyer. Si la communication se fait avec un autre ordinateur où XLogo est lancé, cet ordinateur répondra OK une fois l'opération terminée. Il est également possible de dialoguer avec un robot muni d'une interface réseau, la réponse pourra être différente à ce moment.

  Exemple : `tortue` veut envoyer à `lievre` la séquence "3.14159 presque le nombre pi".
  `lievre` lance `ecoutetcp` pour se mettre en attente de requête d'ordinateurs du réseau. `tortue` lance alors : `ecris envoietcp "lievre [3.14159 presque le nombre pi]`.

Une petite astuce : Lancer deux fois XLogo sur le même ordinateur.
- Dans la première fenêtre, lancer `ecoutetcp`.
- Dans la seconde, lancer `executetcp "127.0.0.1 [av 100 td 90]`

Vous avez ainsi déplacer la tortue sur l'autre fenêtre ! (éh oui, 127.0.0.1 désigne l'adresse locale donc l'ordinateur lui-même...)

Chapitre 5

Exemples de programmes

5.1 Dessiner des maisons

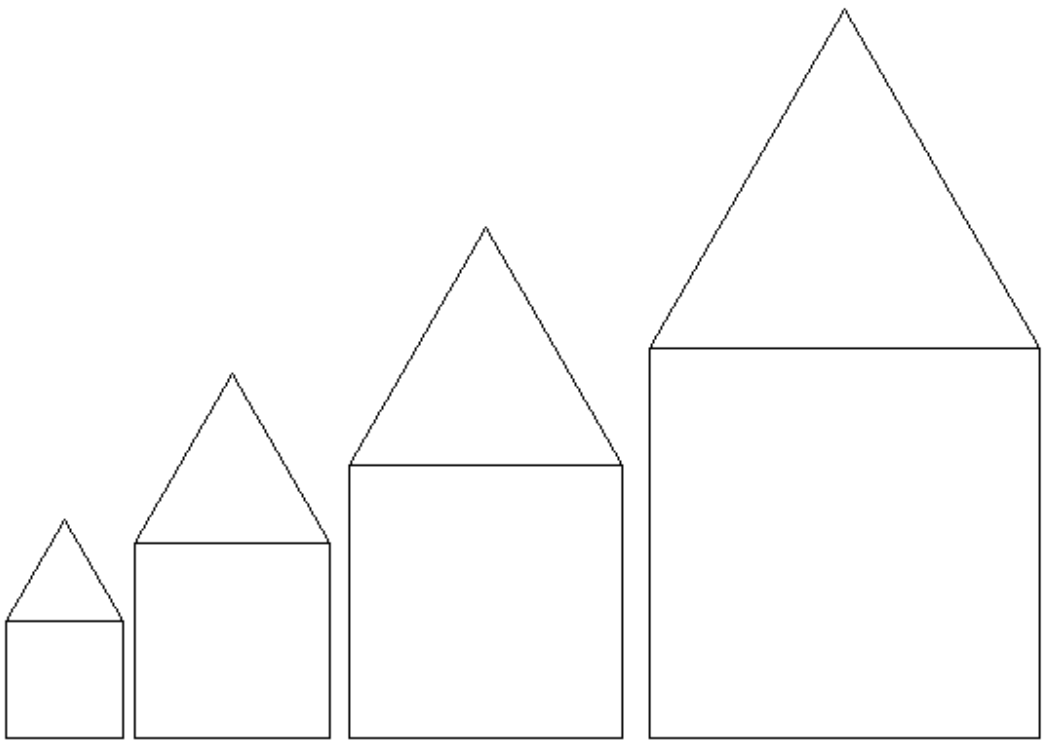


FIG. 5.1 – Maisons

```
pour maison :c
repete 4[av 20*:c td 90]
av 20*:c td 30
repete 3[av 20*:c td 120]
fin

pour dep :c
lc tg 30 re :c*20 td 90 av :c*22 tg 90 bc
```

```
fin

pour mai
ve lc tg 90 av 200 td 90 bc ct
maison 3 dep 3 maison 5 dep 5 maison 7 dep 7 maison 10
fin
```

5.2 Dessiner un rectangle plein

FIG. 5.2 – Rectangle

```
pour rec :lo :la
si :lo=0|:la=0 [stop]
repete 2[av :lo td 90 av :la td 90]
rec :lo-1 :la -1
fin
```

5.3 Factorielle

Rappel : $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

```
pour fac :n
si :n=0[ret 1][ret :n*fac :n-1]
fin

ec fac 5
120
ec fac 6
720
```

5.4 Le flocon (Merci à Georges Noël)

```
pour koch :ordre :lon
si :ordre < 1 | :lon <1 [avance :lon stop]
koch :ordre-1 :lon/3
tg 60
koch :ordre-1 :lon/3
td 120
koch :ordre-1 :lon/3
tg 60
koch :ordre-1 :lon/3
fin
```

```
pour kochflake :ordre :lon
repete 3 [td 120  koch :ordre :lon]
fin

kochflake 5 450
```

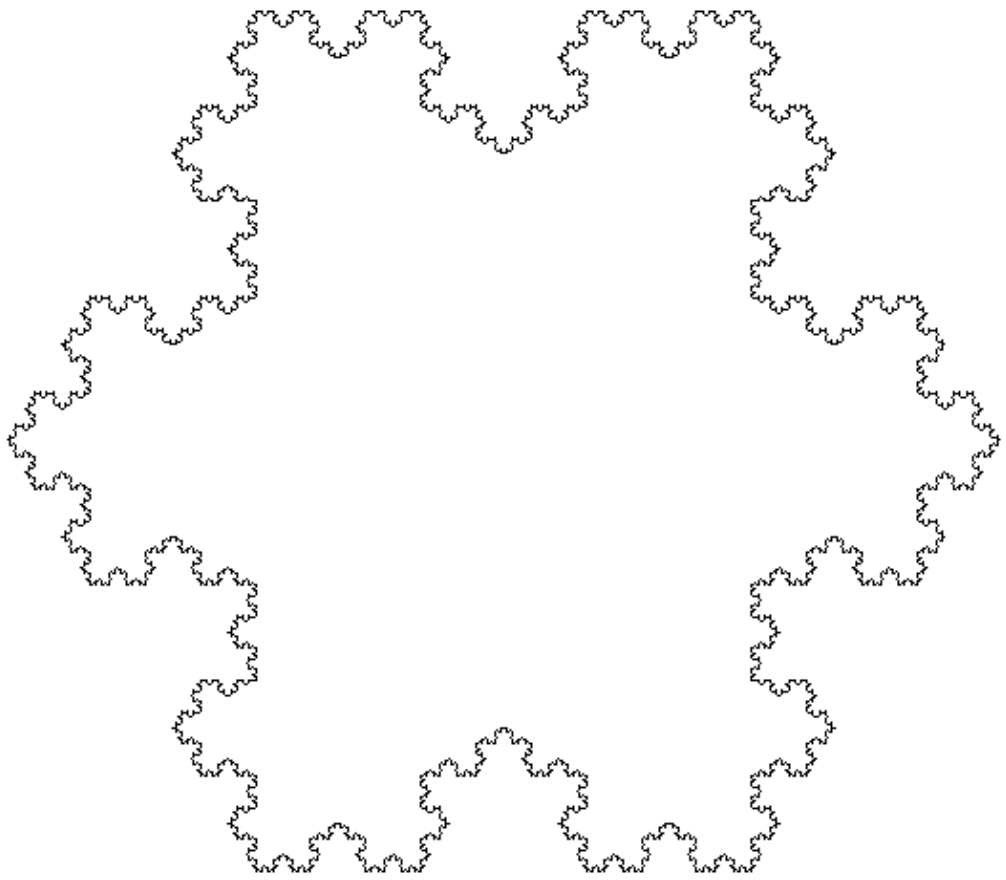


FIG. 5.3 – Le flocon

5.5 Un peu d'écriture...

```
pour ecrire
ct repete 40[av 30 td 9 fcc hasard 7 etiquette [xlogo c'est sympa!]]
fin
```

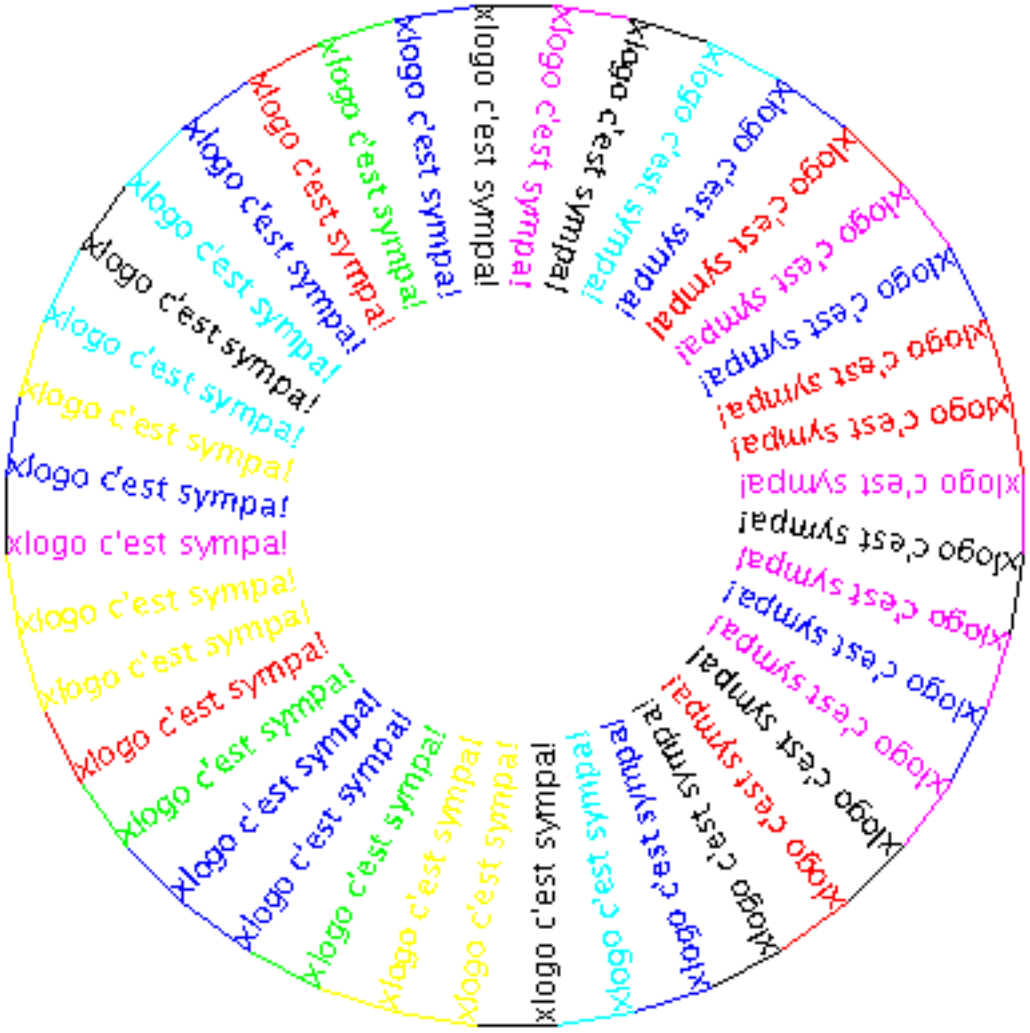


FIG. 5.4 – Xlogo c'est sympa !

5.6 Et de conjugaison...

5.6.1 Première version

```
pour futur :mot
ec ph "je mot :mot "ai
ec ph "tu mot :mot "as
ec ph "il mot :mot "a
ec ph "nous mot :mot "ons
ec ph "vous mot :mot "ez
ec ph "elles mot :mot "ont
fin

futur "parler

je parlerai
tu parleras
il parlera
nous parlerons
vous parlerez
elles parleront
```

5.6.2 Deuxième mouture

```
pour fut :mot
donne "pronoms [je tu il nous vous elles]
donne "terminaisons [ai as a ons ez ont]
donne "i 0
repete 6[donne "i :i+1 ec ph item :i :pronoms mot :mot item :i :terminaisons]
fin

fut "parler

je parlerai
tu parleras
il parlera
nous parlerons
vous parlerez
elles parleront
```

5.6.3 Ou alors : Un peu de récursivité !

```
pour fu :verbe
donne "pronoms [je tu il nous vous elles]
donne "terminaisons [ai as a ons ez ont]
conjugue :verbe :pronoms :terminaisons
fin
```

```
pour conjugue :verbe :pronoms :terminaisons
si vide? :pronoms [stop]
ec ph premier :pronoms mot :verbe premier :terminaisons
conjugue :verbe sp :pronoms sp :terminaisons
fin

fu "parler

je parlerai
tu parleras
il parlera
nous parlerons
vous parlerez
elles parleront
```

5.7 Avec les couleurs

5.7.1 Introduction

Tout d'abord, quelques explications : vous remarquerez que la commande `fcc` prend soit en paramètre une liste soit un nombre. Ici, nous allons nous intéresser au codage [r g b] Quésako ? ? ? ?
Chaque couleur dans XLOGO est codée sur trois composantes, la rouge (red), la verte (green) et la bleue (blue) d'où le nom de ce codage r g b. Les trois nombres donnés dans la liste à la primitive `fcc` représentent donc dans l'ordre la composante rouge, puis la verte, puis la bleue d'une couleur. Ce codage n'étant pas très intuitif, vous pouvez avoir un aperçu de la couleur obtenue selon le codage à l'aide la boîte de dialogue Options—-> Choisir la couleur du crayon.
A l'aide de ce codage, il est très facile d'effectuer des transformations sur une image. Par exemple, si l'on veut mettre en niveau de gris une photo couleur, on change la couleur de chaque pixel de l'image en la moyenne des trois composantes [r g b]. Je m'explique : imaginons qu'un point de l'image est pour couleur [0 100 80]. On calcule la moyenne de ces 3 nombres : $\frac{0+100+80}{3} = 60$. On réaffecte alors à ce pixel la couleur [60 60 60]. Cette opération doit être effectuée sur chaque pixel de l'image

5.7.2 Passons à la pratique

Nous allons transformer en niveau de gris une image de 100 pixels sur 100 pixels, il y aura donc $100 \times 100 = 10000$ pixels à modifier. Vous pouvez trouver l'image utilisée dans l'exemple suivant à l'adresse :
http ://xlogo.tuxfamily.org/images/transfo.png
Voici comment nous allons procéder : nous allons afficher l'image avec le coin supérieur gauche en [0 0]. Ensuite, la tortue va décrire les 100 premiers pixels de la première ligne puis les 100 premiers de la deuxième etc... A chaque fois, on rapatrie la couleur du pixel avec `trouvecouleur` puis on modifie la couleur en faisant la moyenne des trois composantes [r g b]. Voici le code correspondant :
(N'oubliez pas de changer le chemin dans la procédure transfo)

```
pour li :liste
# renvoie la moyenne des trois nombres [r g b]
donne "r premier :liste
donne "liste sp :liste
donne "g premier :liste
donne "liste sp :liste
donne "b premier :liste
donne "b arrondi (:r+:g+:b)/3
retourne ph liste :b :b :b
fin

pour niveaudegris :c
si :y=-100 [stop]
si :c=100 [donne "c 0 donne "y  :y-1]
# On prend pour couleur de crayon la couleur "moyennée" du pixel suivant
fcc li tc liste :c :y
# On repasse en niveau de gris
point liste :c :y
niveaudegris :c+1
fin

pour transfo
# Vous devez changer le chemin vers l'image transfo.png
# Ex: frep "c:\\windows chargeimage "transfo.png

ve ct frep "/home/loic chargeimage "transfo.png donne "y 0 niveaudegris 0
fin
```

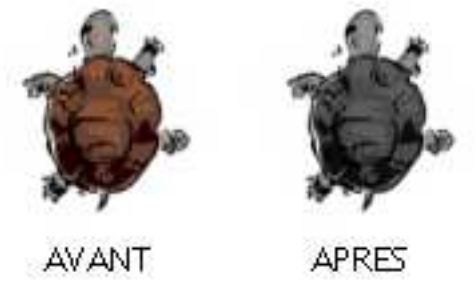


FIG. 5.5 – XLOGO fait de la retouche d'images....

5.7.3 Et si on la voyait en négatif ? ?

Pour changer une image en négaitf, même procédé sauf qu'au lieu de moyenner les nombres r g b, on les remplace par leur complément à 255. Ex : Un pixel a pour couleur [2 100 200], on remplace sa couleur par : [253 155 55]

Seul la procédure `li` change par rapport à tout de suite :

```
pour niveaudegris :c
si :y=-100 [stop]
si :c=100 [donne "c 0 donne "y  :y-1]
fcc li tc liste :c :y
point liste :c :y
niveaudegris :c+1
fin

pour transfo
# Vous devez changer le chemin vers l'image transfo.png
# Ex: frep "c:\\windows\\Bureau chargeimage "transfo.png
ct ve frep "/home/loic ci "transfo.png donne "y 0 niveaudegris 0
fin

pour li :liste
donne "r premier :liste
donne "liste sp :liste
donne "g premier :liste
donne "liste sp :liste
donne "b premier :liste
retourne ph liste 255-:r 255-:g 255-:b
fin
```

FIG. 5.6 – XLOGO se prend pour LE GIMP ...(prétentieux o :))

5.8 Un bel exemple d'utilisation des listes (Merci Olivier SC)

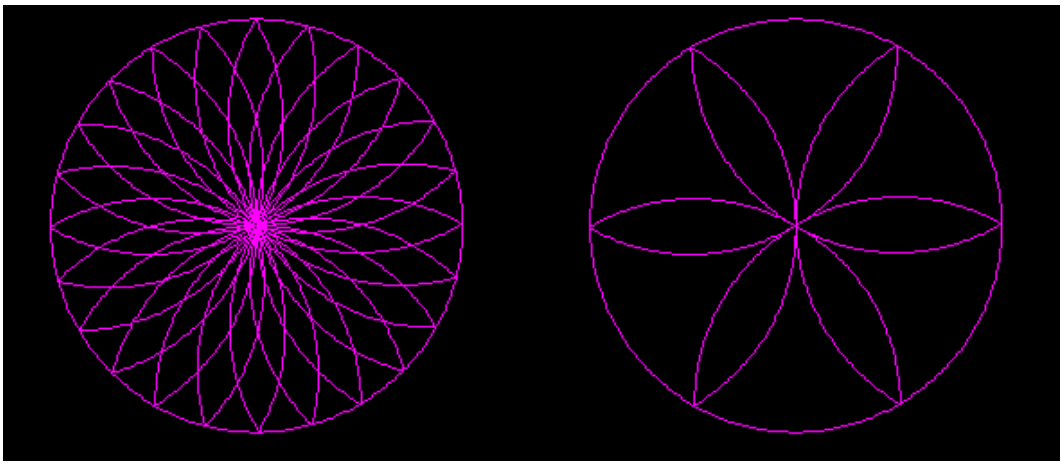
Je vous laisse apprécier ce magnifique programme :

```
pour inversem :m
si vide? :m [retourne "]
retourne mot dernier :m inversem sd :m
fin
```

```
pour palindrome :m
SI EGAL? :M INVERSEM :M [RETOURNE "VRAI] [RETOURNE "FAUX]
fin

pour palin :n
SI PALINDROME :N [ECRIS :N STOP]
ECRIS PH PH PH PH :N "PLUS INVERSEM :N "EGAL SOMME :N INVERSEM :N
PALIN :N + INVERSEM :N
fin

palin 78
78 PLUS 87 EGAL 165
165 PLUS 561 EGAL 726
726 PLUS 627 EGAL 1353
1353 PLUS 3531 EGAL 4884
4884
```

5.9 Une belle rosace

```
pour rosace
repete 6[ repete 60[av 2 td 1] td 60 repete 120 [av 2 td 1] td 60]
fin

pour belle_rosace
rosace
repete 30[av 2 td 1]
rosace
repete 15[av 2 td 1]
rosace
repete 30[av 2 td 1]
rosace
fin

fcfg 0 ve fcc 5 ct rosace lc fpos [-300 0] bc fixecap 0 belle_rosace
```



Chapitre 6

Désinstallation-mise à jour

6.1 Désinstallation

Pour désinstaller XLogo, il suffit de supprimer le fichier XLogo.jar et le fichier de démrrage .xlogo (il est situé dans votre répertoire utilisateur c'est à dire /home/votre_login pour les linuxiens ou c :\windows\.xlogo

6.2 Mise à jour

Pour toute nouvelle version et les corrections de bugs, se rendre de temps en temps sur le site de XLogo c'est à dire http ://xlogo.tuxfamily.org Ne pas hésiter à écrire en cas de problème d'installation ou d'utilisation. Toute suggestion est la bienvenue.

Chapitre 7

Foire aux questions - Astuces - trucs à connaître

7.1 J'ai beau effacer une procédure dans l'éditeur, elle réapparaît tout le temps !

Lorsqu'on sort de l'éditeur, celui-ci se contente juste de sauver ou de mettre à jour le contenu de l'éditeur. Le seul moyen d'effacer une procédure dans XLogo est d'utiliser la primitive `effacenom` ou `efn`.
Exemple : `effacenom "toto` ⟶ efface la procédure `toto`.

7.2 J'utilise la version en espéranto mais je ne peux écrire les caractères spéciaux !

Lorsque vous tapez dans la ligne de commande ou l'éditeur, si vous faites un clic avec le bouton droit de la souris, apparait un menu déroulant. Dans ce menu, figurent les traditionnels fonctions d'édition (copier, couper, coller) et les caractères spéciaux de l'espéranto lorsque ce langage est sélectionné.

7.3 Dans l'onglet Son de la boîte de dialogue Préférences, aucun instrument n'est disponible.

Désolé, ce problème est connu. Il est dû à la machine virtuelle java. Ce problème est totalement aléatoire. Par exemple, chez moi, j'ai un ordinateur qui tourne sous linux et win98. Sous win98, la liste n'apparait pas, sous linux si ! ! Pourtant j'utilise le même JRE et je n'ai aucun problème matériel. D'une version d'un JRE à l'autre, ce comportement peut changer.

7.4 J'ai des problèmes de rafraichissement d'écran lorsque la tortue dessine !

Problème connu également et typique des JRE> 1.4.2. J'essaierai d'y remédier dans le futur, je peux peut-être faire quelque chose. Deux solutions pour le moment :

- minimiser la fenêtre et la réagrandir.
- Utiliser le JRE 1.4.1_07 proposé sur le site.

7.5 Comment faire pour taper rapidement une commande déjà utilisée au préalable ?

- Première méthode : avec la souris, cliquer dans la zone d'historique sur la ligne désirée, elle réapparaîtra immédiatement dans la ligne de commande.
- Deuxième méthode : avec le clavier, les flèches Haut et Bas permettent de naviguer dans la liste des dernières commandes tapées(Très pratique).

7.6 Comment peut-on vous aider ?

- En reportant tout bug constaté. Si vous êtes capable de reproduire systématiquement un problème constaté, c'est encore mieux.
- Vos suggestions en vue de l'amélioration sont toujours les bienvenues.
- En aidant aux traductions : en particulier l'anglais...
- Un petit encouragement fait toujours du bien !

Index

absolue abs, 17
ajoute, 19
ajouteligneflux, 26
animation, 13
arc, 12
arccosinus, acos, 17
arcsinus, asin, 17
arctangente, atan, 17
arrondi, 17
attends, 38
avance, av, 12

baissecrayon, bc, 13
baissecrayon ?, bc ?, 21
blanc, 11
bleu, 11
bleufonce, 11

cachetortue, ct, 13
cap, 13
caractere,car, 20
catalogue, cat, 25
cercle, 12
changedossier, cd, 25
chargeimage, ci, 25
chattcp, 40
choix, 19
chose, 24
clos, 14
compte, 20
compteur, 33
cosinus, cos, 17
couleurcrayon, cc, 14
couleurfond, cf, 14
couleurtexte, ctexte, 16
cyan, 11

date, 38
debuttemps, 38
definis, def, 24
dernier, der, 19
dessine, de, 13

difference, 17
distance, 14
div, divise, 17
donne, 24
donnelocale, 24

ec, ecris, 16
ecoutetcp, 39
ecrisligneflux, 26
effacenom, efn, 24
effacenoms, efns, 24
effacesequence, efseq, 31
effacevariable, efv, 24
egal ?, 21
enleve, 19
enroule, enr, 14
entier ?, 21
envoietcp, 40
et, 18
etiquette, 12
execute, exec, 24
executetcp, 39

faux, 21
fenetre, fen, 14
fermeflux, 26
fin, 22
finflux ?, 26
finstrument, finstr, 31
fintemps ?, 38
fixecap, 12
fixecouleurcrayon, fcc, 13
fixecouleurfond, fcfg, 13
fixecouleurtexte, fct, 16
fixeforme, fforme, 14
fixeindexsequence, findseq, 31
fixenompolice, fnp, 14
fixenompolicetexte, fnpt, 16
fixeposition, fpos, 12
fixerepertoire, frep, 25
fixeseparation, fsep, 15
fixestyle, fsty, 16

fixetaillecrayon, ftc, 14
fixetaillepolice, ftp, 14
fixetaillepolicetexte, ftpt, 16
fixetortue, ftortue, 31
fixex, 12
fixexy, 12
fixey, 12
forme, 14

gomme, go, 13
gris, 11
grisclair, 11

hasard, 17
heure, 38

imts, 24
indexsequence, indseq, 31
instrument, instr, 31
inverse, 19
inversecrayon, ic, 13
item, 19

jaune, 11
joue, 31

levecrayon,lc, 13
lis, 35
liscar, 35
liscarflux, 26
lisligneflux, 26
lissouris, 36
liste, 19
liste ?, 21
listeflux, 26
listevariables, 24
locale, 24
log10, 17

magenta, 11
marron, 11
membre, 21
membre ?, 21
message, msg, 15
metsdernier, md, 19
metspremier, mp, 19
moins, 17
montretortue, mt, 13
mot, 19
mot ?, 21

nettoie, 13
noir, 11
nombre ?, 21
nompolice, np, 15
nompolicetexte, npt, 16
non, 18

orange, 11
origine, 12
ou, 18
ouvreflux, 25

phrase, ph, 19
pi, 17
point, 13
pos, 13
possouris, 36
pour, 22
precede ?, 21
premier, prem, 19
primitive ?, prim ?, 21
procedure ?, proc ?, 21
produit, 17
puissance, 17

quotient, 17

racine, rac, 17
rafraichis, 13
ramene, 25
recule, re, 12
remplace, 19
remplis, 28
rempliszone, 28
repertoire, rep, 25
repete, 33
repetepour, 33
reste, 17
retourne, 30
rose, 11
rouge, 11
rougefonce, 11

saufdernier, sd, 19
saufpremier, sp, 19
sauve, 25
sauved, 25
separation,sep, 15
sequence, seq, 31
si, 22
sinus, sin, 17
somme, 17

souris ?, 36
stop, 30
stoptout, 30
sty, style, 16

taillepolice, tp, 14
taillepolicetexte, tpt, 16
tangente, tan, 17
tantque, 34
tape, 16
temps, 38
tortue, 31
tortues, 31
touche ?, 35
tournedroite, td, 12
tournegauche, tg, 12
trace, 23
tronque, 17
trouvecouleur, tc, 14
tuetortue, 31

unicode, 20

vers, 13
vert, 11
vertfonce, 11
vide ?, 21
videecran, ve, 13
violet, 11
visible ?, 21
vrai, 21
vt, videtexte, 16